Diktat Kuliah
IK-311 Algoritma dan Pemrograman II

Oleh:
Rosa Ariani Sukamto

Program Ilmu Komputer
Universitas Pendidikan Indonesia
2010

Thanks to:

Allah, my husband, my daughter, and my students to become my inspiration sources. Thank you all.

- Rosa A. S.

Kata Pengantar

Puji syukur atas kehadirat Allah SWT yang telah melimpahkan rahmatNya kepada penulis sehingga penulis dapat menyelesaikan buku yang berjudul **Diktat Algoritma dan Pemrograman II** untuk Ilmu Komputer Universitas Pendidikan Indonesia .

Algoritma dan Pemrograman adalah jantung dari pemahaman mengenai pemrograman dan merupakan fondasi awal bagi ilmu komputer. Buku ini dibuat dengan tulus bertujuan agar para mahasiswa dapat lebih mudah memahami bagaimana membuat algoritma dan memetakannya dalam bahasa pemrograman (Amin). Buku ini dibuat sebagai kelanjutan buku diktat Algoritma dan Pemrograman I. Sebaiknya pembaca sudah memahami Algoritma dan Pemrograman I untuk dapat mengerti isi buku ini.

Seperti slogan dalam sebuah iklan mobil "*practice make perfect*". Jika pembaca ingin menjadi seorang yang berkecimpung di dunia teknologi informasi yang hebat maka pembaca harus banyak berlatih dan memahami konsep. Kesabaran sangat dibutuhkan untuk menjadi sukses. Karena kesuksesan adalah keberhasilan yang terencana dimana selalu mampu bangkit dari segala rintangan. Praktikum juga sangat memegang peranan penting dalam mata kuliah ini.

Penulis merasa masih banyak sekali kekurangan dalam peyusunan buku ini. Akhirnya penulis mengucapkan terima kasih kepada semua pihak yang tidak dapat penulis sebutkan satu persatu, yang telah membantu penyusunan buku ini.

Segala saran, kritik, dan pertanyaan dapat dikirimkan melalui email penulis atau disampaikan langsung kepada penulis.

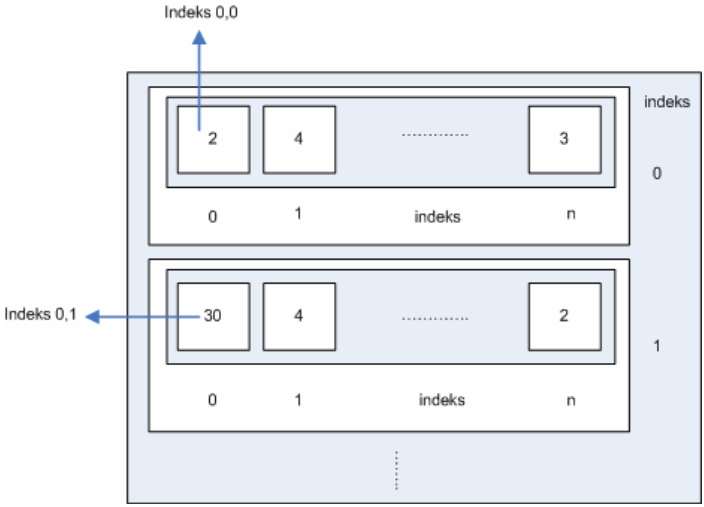
Bandung, Februari 2010

Penulis

Daftar Isi

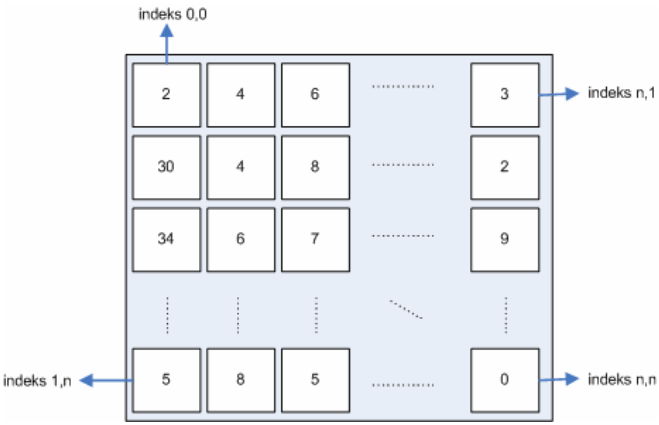
1. Matrik (Array 2 Dimensi)

Matrik atau *array* dua dimensi adalah *array* yang memiliki dua atau lebih kolom dengan banyak baris, atau dua atau lebih baris dengan banyak kolom, bergantung pada bagaimana kita mengilustrasikannya dalam pikiran kita, untuk lebih jelasnya mengenai *array* dua dimensi dapat dilihat pada gambar 1.



Gambar 1 Ilustrasi Larik Dua Dimensi

sebenarnya *array* dua dimensi adalah sebuah *array* yang ada di di dalam *array*, misalnya ada sebuah *array* dua dimensi dengan ukuran 2 x 2 maka pada sebuah tempat sel pertama akan ada sebuah *array* satu dimensi di dalamnya sehingga gambaran secara lebih mudah untuk merepresentasikan *array* dua dimensi dapat dilihat pada gambar 2.



Gambar 2 Representasi Larik Dua Dimensi

Deklarasi array dua dimensi adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
nama_array : array [1…nBaris, 1…nKolom] of tipe_data	type nama_tipe_array = array[1..jumlah_baris, 1..jumlah_kolom] of tipe_data; var nama_array : nama_tipe_array;	tipe_data nama_array[jumlah_baris] [jumlah_kolom];
tabInt : array [1…20, 1…10] of integer	type tabel = array [1..20, 1..10] of integer; var tabInt : tabel;	int tabInt[20][10];

sedangkan cara mengaksesnya adalah sebagai berikut :

Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
mengisi nilai array	nama_array$_{baris}$, kolom <-	nama_array[baris, kolom] := nilai;	nama_array[baris] [kolom] = nilai;

Keterangan	Bahasa Algoritmik	Bahasa Pascal	Bahasa C
dengan indeks tertentu	nilai		
	$tabInt_{1,\ 1}$ <- 5	tabInt[1, 1] := 5;	tabInt[0][0] = 5;
mengakses nilai array dengan indeks tertentu	$nama\_array_{baris\ ,\ kolom}$	nama_array[baris, kolom];	nama_array[baris] [kolom];
	$tabInt_{1,\ 1}$	tabInt[1, 1];	tabInt[0][0];

1.1. Proses-proses Pada Matriks

1.1.1. Mengisi Matriks

Pengisian Matriks dapat dilakukan dengan menggunakan pengulangan `for`, berikut adalah algoritma mengisi matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat matriks yang akan diisi	matriks : array [1..4, 1..4] of integer
Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks	penghitung_baris : integer penghitung_kolom : integer

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks
	for penghitung_baris <- 1 to 4 do
	
	for penghitung_kolom <- to 4 do
	
	input($matriks_{penghitung\_baris,\ penghitung\_kolom}$)
	
	{*end for*}
	
	{*end for*}

Bahasa Manusia	Bahasa Algoritmik
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom	

1.1.2. Menampilkan Elemen Matriks

Menampilkan elemen matriks dapat dilakukan dengan menggunakan pengulangan `for`, matriks harus sudah diisi agar dapat ditampilkan isinya, berikut adalah algoritma menampilkan isi matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat matriks yang akan diisi	`matriks : array [1..4, 1..4] of integer`
Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks	`penghitung_baris : integer` `penghitung_kolom : integer`

Melakukan pengisian matriks
Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks
 	
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom
	`for penghitung_baris <- 1 to 4 do`
	
	`for penghitung_kolom <- to 4 do`
	
	`input(matriks`$_{penghitung\_baris,\ penghitung\_kolom}$`)`
	
	`{end for}`
	`{end for}`

Melakukan penampilan matriks ke layar
 	
Melakukan pengulangan per baris
	`for penghitung_baris <- 1 to 4 do`
	
	`for penghitung_kolom <- to 4 do`

Bahasa Manusia	Bahasa Algoritmik
Melakukan pengulangan per kolom menampilkan isi matriks ke layar menampilkan ganti baris	output(matriks$_{penghitung_baris, penghitung_kolom}$, " ") {*end for*} output("\n") {*end for*}

1.1.3. Salin Matriks

Salin Matriks dapat dilakukan dengan menggunakan pengulangan `for`, salin matriks berarti menyalin isi matriks satu ke matriks lainnya yang memiliki dimensi/ukuran yang sama, berikut adalah algoritma salin matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat matriks yang akan diisi dan matriks yang akan diisi dengan hasil salinan matriks yang telah diisi sebelumnya	matriks1 : array [1..4, 1..4] of integer matriks2 : array [1..4, 1..4] of integer
Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks	penghitung_baris : integer penghitung_kolom : integer

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks
	for penghitung_baris <- 1 to 4 do
	for penghitung_kolom <- to 4 do
	input(matriks1$_{penghitung_baris, penghitung_kolom}$)
	{*end for*}
	{*end for*}

Bahasa Manusia	Bahasa Algoritmik
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom	

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
mengisi matriks2 dengan isi dari matriks1
 	
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom
	for penghitung_baris <- 1 to 4 do
	
	for penghitung_kolom <- to 4 do
	
	matriks2penghitung_baris, penghitung_kolom <- matriks1penghitung_baris, penghitung_kolom
	
	{end for}
	{end for}

1.1.4. Menjumlahkan Dua Buah Matriks

Penjumlahan dua buah matriks dapat dilakukan dengan menggunakan pengulangan `for`, berikut adalah algoritma penjulahan dua buah matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat 2 matriks yang akan diisi dan dijumlahkan, dan satu matriks untuk menyimpan hasil penjumlahan	matriks1 : array [1..4, 1..4] of integer matriks2 : array [1..4, 1..4] of integer matriks3 : array [1..4, 1..4] of integer
Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks	penghitung_baris : integer penghitung_kolom : integer

Melakukan pengulangan per baris
 	
Melakukan pengulangan per
	for penghitung_baris <- 1 to 4 do
	
	for penghitung_kolom <- to 4 do

Bahasa Manusia	Bahasa Algoritmik
kolom meminta masukan *user* /pemakai program untuk memasukkan elemen matriks1 Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom	input(matriks1$_{penghitung_baris, penghitung_kolom}$) {*end for*} {*end for*}

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks2
 	
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom
	for penghitung_baris <- 1 to 4 do
	for penghitung_kolom <- to 4 do
	input(matriks2$_{penghitung_baris, penghitung_kolom}$)
	{*end for*}
	{*end for*}

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
mengisi elemen matriks3 dengan hasil pertambahan matriks1 dan matriks2
	for penghitung_baris <- 1 to 4 do
	for penghitung_kolom <- to 4 do
	matriks3$_{penghitung_baris, penghitung_kolom}$ <- matriks1$_{penghitung_baris, penghitung_kolom}$ + matriks2$_{penghitung_baris, penghitung_kolom}$

Bahasa Manusia	Bahasa Algoritmik
	{*end for*} {*end for*}

1.1.5. Mengalikan Dua Buah Matriks

Pengalian dua buah matriks dapat dilakukan dengan menggunakan pengulangan `for`, berikut adalah algoritma pengalian dua buah matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat 2 matriks yang akan diisi dan dijumlahkan, dan satu matriks untuk menyimpan hasil penjumlahan	matriks1 : array [1..4, 1..4] of integer matriks2 : array [1..4, 1..4] of integer matriks3 : array [1..4, 1..4] of integer
Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks	penghitung_baris : integer penghitung_kolom : integer penghitung_jumlah_kali : integer

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks1
 	
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom
	for penghitung_baris <- 1 to 4 do
	
	for penghitung_kolom <- to 4 do
	
	input(matriks1penghitung_baris, penghitung_kolom)
	
	{*end for*}
	
	{*end for*}

Melakukan pengulangan per baris	for penghitung_baris <- 1 to 4 do

Bahasa Manusia	Bahasa Algoritmik
Melakukan pengulangan per kolom meminta masukan *user* /pemakai program untuk memasukkan elemen matriks2 Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom	for penghitung_kolom <- to 4 do input(matriks2penghitung_baris, penghitung_kolom) {end for} {end for}

Melakukan pengulangan per baris (dari sudut pandang matriks3)
 	
Melakukan pengulangan per kolom (dari sudut pandang matriks3)
 	
inisialisasi elemen matriks3 dengan 0
 	
melakukan pengulangan per penjumlahan elemen matriks1 dan matriks2 yang sudah dikalikan
 	
mengisi elemen matriks3 dengan hasil pengalian matriks1 dan matriks2
	for penghitung_baris <- 1 to 4 do
	
	for penghitung_kolom <- to 4 do
	
	matriks3penghitung_baris, penghitung_kolom <- 0
	
	for penghitung_jumlah_kali <- 1 to 4 do
	
	matriks3penghitung_baris, penghitung_kolom <- matriks3penghitung_baris, penghitung_kolom + (matriks1penghitung_baris, penghitung_jumlah_kali * matriks2 penghitung_jumlah_kali, penghitung_kolom)
	
	{end for}

Bahasa Manusia	Bahasa Algoritmik
	{*end for*}

1.1.6. Mencari Elemen Matriks Tertentu

Pengisian matriks dapat dilakukan dengan menggunakan pengulangan `for`, berikut adalah algoritma mengisi matriks:

Bahasa Manusia	Bahasa Algoritmik
Membuat matriks yang akan diisi	`matriks : array [1..4, 1..4] of integer`

Membuat kotak penghitung baris dan penghitung kolom untuk melakukan pengulangan pengisian matriks, sebuah kotak untuk memasukkan nilai yang dicari di dalam matriks, sebuah kotak untuk status nilai sudah ketemu atau belum	`penghitung_baris : integer`
	`penghitung_kolom : integer`
	`yang_dicari : integer`
	`ketemu : boolean`

Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
meminta masukan *user* /pemakai program untuk memasukkan elemen matriks
 	
Secara logika sebuah matriks diisi per baris dan setiap barisnya diisi per kolom
	`for penghitung_baris <- 1 to 4 do`
	
	`for penghitung_kolom <- to 4 do`
	
	`input(matriks`$_{\text{penghitung\_baris, penghitung\_kolom}}$`)`
	
	{*end for*}
	
	{*end for*}

Meminta masukan *user* /pemakai program untuk memasukkan nilai yang dicari dan dimasukkan di dalam kotak `yang_dicari`	`input(yang_dicari)`

Bahasa Manusia	Bahasa Algoritmik

Melakukan pengulangan untuk melakukan pencarian pada elemen matriks dengan logika sebagai berikut:
 	
inisialisasi `ketemu`, `penghitung_baris`, dan `penghitung_kolom`
 	
Melakukan pengulangan per baris
 	
Melakukan pengulangan per kolom
 	
jika isi elemen sama dengan yang_dicari maka set ketemu dengan true agar pengulangan berhenti
 	
lainnya
iterasi penghitung_kolom
 	
iterasi penghitung_baris
	`ketemu <- false`
	`penghitung_baris <- 1`
	`penghitung_kolom <- 1`
	`while (ketemu = false) and (penghitung_baris <= 4) do`
	
	`while (ketemu = false) and (penghitung_kolom <= 4) do`
	
	`if matriks`$_{penghitung\_baris,\ penghitung\_kolom}$ `= yang_dicari then`
	
	`ketemu <- true`
	
	{*end if*}
	`else`
	`penghitung_kolom <- penghitung_kolom + 1`
	
	{*end else*}
	
	{*end while*}
	
	`penghitung_baris <- penghitung_baris + 1`
	
	{*end while*}

Menampilkan apakah `yang_dicari` ketemu atau tidak	`if ketemu = true then`
	`output("bilangan yang dicari ada di dalam matriks")`
	{*end if*}
	`else`
	`output("bilangan yang dicari tidak ada di matriks")`
	{*end else*}

2. Rekursif

2.1. Pengertian Rekursif

Rekursif adalah proses memanggil dirinya sendiri yang biasanya dilakukan oleh fungsi atau prosedur pada pemrograman prosedural. Rekursif akan terus berjalan sampai kondisi berhenti terpenuhi, oleh karena itu dalam sebuah rekursiff perlu adanya blok-blok kode sebagai berikut :

- **Basis**
  Basis merupakan kode yang menjadi titik berhenti dari sebuah proses rekursif karena proses rekursif akan terus berjalan berputar terus memanggil dirinya sendiri sampai sebuah kondisi basis terpenuhi, oleh karena itu basis sangat penting dalam sebuah proses rekursif, karena tanpa basis sebuah proses rekursif akan terus dijalankan tanpa henti.
- **Rekursif**
  Rekursif merupakan kode dimana sebuah blok program (prosedur, fungsi, atau metode) memanggil dirinya sendiri, misalnya ada sebuah fungsi hitung hasil faktorial, dimana di dalamnya blok kode fungsi itu memanggil dirinya sendiri.

2.2. Proses Rekursif

Contoh proses rekursif yang paling sederhana, yaitu menuliskan sesuatu ke layar, misalnya sebagai berikut :

```
procedure tulis(input : n : integer)
  counter : integer
  counter <- n
  if  counter > 0 then
    output("proses rekursiff dengan nilai counter : ", counter)
    counter <- counter - 1
    tulis(counter)
 {end if}
{end procedure}
```

dengan penjelasan langkah-langkah prosedur sebagai berikut :

Penjelasan	Bahasa Algoritmik
Mendeklarasikan prosedur yang mengandung rekursif dengan masukan bilangan bulat	`procedure tulis(input : integer n)`
Mendeklarasikan variabel bilangan bulat yang nantinya nilainya akan dimanipulasi sebagai basis yang nilainya diisi dengan nilai masukan n	`counter : integer` `counter <- n`
Kondisi basis yaitu selama nilai variabel counter lebih besar dari nol, maka proses di dalam blok precabangan if terus dikerjakan, dan akan berhenti bila nilai variabel counter adalah nol	`if counter > 0 then`

Proses rekursif beserta inisialisasinya dimana prosedur tulis akan dipanggil kembali dengan nilai masukan nilai sebelumnya dikurangi satu	`output("proses rekursiff dengan nilai counter : ", counter)`
	`counter <- counter - 1`
	`tulis(counter)`

Memanggil prosedur tulis pada algoritma utama	`{algoritma utama}`
	`tulis(10)`
	`{end algoritma utama}`

Setelah dilakukan kompilasi dan eksekusi program maka akan menghasilkan keluaran sebagai berikut :

```
proses rekursif dengan nilai counter : 10
proses rekursif dengan nilai counter : 9
proses rekursif dengan nilai counter : 8
proses rekursif dengan nilai counter : 7
proses rekursif dengan nilai counter : 6
proses rekursif dengan nilai counter : 5
proses rekursif dengan nilai counter : 4
proses rekursif dengan nilai counter : 3
proses rekursif dengan nilai counter : 2
proses rekursif dengan nilai counter : 1
```

Agar lebih jelas dapat ditelusuri proses rekursif di atas sebagai berikut :

- **Proses rekursif ke-1**
  Variabel counter bernilai sepuluh, jalannya program masuk ke dalam blok percabangan if karena nilai counter adalah sepuluh lebih besar dari nol, proses pada percabangan if dijalankan yaitu menuliskan nilai variabel counter ke layar, nilai variabel counter dikurangi satu, kemudian prosedur tulis dipanggil kembali dengan masukan bernilai sembilan.
- **Proses rekursif ke-2**
  Variabel counter bernilai sembilan, jalannya program masuk ke dalam blok percabangan if karena nilai counter adalah sembilan lebih besar dari nol, proses pada percabangan if dijalankan yaitu menuliskan nilai variabel counter ke layar, nilai variabel counter dikurangi satu, kemudian prosedur tulis dipanggil kembali dengan masukan bernilai delapan.

- **Proses rekursif ke-10**
  Variabel counter bernilai satu, jalannya program masuk ke dalam blok percabangan if karena nilai counter adalah satu lebih besar dari nol, proses pada percabangan if dijalankan yaitu menuliskan nilai variabel counter ke layar, nilai variabel counter dikurangi satu, kemudian prosedur tulis dipanggil kembali dengan masukan bernilai nol.
- **Proses rekursif ke-11**
  Variabel counter bernilai nol, jalannya program tidak masuk ke dalam blok percabangan if karena nilai counter adalah nol, sehingga prosedur tulis tidak dipanggil kembali dan proses rekursif berhenti.

Cara berpikir proses rekursif memang agak sulit dipahami oleh pemula, namun sebagai seorang *programmer* setidaknya dapat mengerti apa yang dimaksud dengan rekursif. Biasanya sebuah proses rekursif dilakukan dengan alur mundur, dimana nilai yang menjadi basis rekursif dikurangi seiring dengan dipanggilnya kembali fungsi rekursif. Rekursif dapat dilakukan secara langsung maupun tak langsung, rekursif langsung adalah jika sebuah fungsi, prosedur, atau metode memanggil dirinya sendiri di dalam bloknya, sedangkan rekursif tak langsung adalah jika sebuah fungsi atau prosedur memanggil fungsi atau prosedur lain yang memanggil dirinya.

Untuk lebih memahami lagi tentang rekursif, berikut contoh penggunaan rekursif dalam menghitung hasil faktorial dari suatu bilangan, dan sebagai perbandingan dapat dilihat cara menghitung faktorial dengan menggunakan perulangan pada bab yang membahas perulangan sebelumnya. Hasil faktorial dari suatu bilangan n adalah 1 x 2 x .....x n sehingga untuk menghitung hasil faktorial dapat dibuat langkah-langkah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik
Mendeklarasikan sebuah prosedur faktorial dengan masukan sebuah bilangan bulat yang akan dihitung nilai faktorialnya dan variabel hasil yang akan menyimpan hasil perhitungan	`procedure faktorial(input : n : integer, input/output: hasil : integer)`
Membuat kondisi basis dengan percabangan if dimana jika nilai variabel yang menjadi basis masih memenuhi syarat berjalannya rekursif maka proses rekursif akan dijalankan (angka 1 tidak dimasukkan karena semua bilangan dikalikan 1 adalah bilangan itu sendiri)	`if n > 1 then`

Membuat inisialisasi proses rekursif di dalam blok percabangan if yaitu mengalikan hasil kali sebelumnya dengan bilangan bulat masukan, menuliskan hasil perkalian ke layar, kemudian mengurangi nilai bilangan masukan dengan satu, dan melakukan proses rekursif	`hasil <- hasil * n`
	`output("rekursif dengan nilai hasil : ", hasil, " nilai n : ", n)`
	`n <- n - 1`
	`faktorial(n, hasil)`

Memanggil prosedur rekursif pada algoritma utama	`{algoritma utama}`
	
	`hasil : integer`
	`hasil <- 1`
	`faktorial(10, hasil)`
	
	`{end algoritma utama}`

Setelah dilakukan kompilasi dan eksekusi program maka akan menghasilkan keluaran sebagai berikut :

```
rekursif dengan nilai hasil : 10, nilai n : 10
rekursif dengan nilai hasil : 90, nilai n : 9
rekursif dengan nilai hasil : 720, nilai n : 8
rekursif dengan nilai hasil : 5040, nilai n : 7
rekursif dengan nilai hasil : 30240, nilai n : 6
rekursif dengan nilai hasil : 151200, nilai n : 5
rekursif dengan nilai hasil : 604800, nilai n : 4
rekursif dengan nilai hasil : 1814400, nilai n : 3
rekursif dengan nilai hasil : 3628800, nilai n : 2
```

Proses rekursif sebaiknya dihindari karena menyebabkan lambatnya jalannya program, sebaiknya diganti dengan menggunakan perulangan jika memungkinkan, tetapi jika tidak dapat, bisa tetap digunakan.

3. Pengurutan (Sorting)

Pengurutan data atau *sorting* merupakan hal yang penting dalam kehidupan nyata untuk memudahkan pengelolaan data. Pengurutan dapat dilakukan dengan urutan menaik (*ascending*) atau dengan urutan menurun (*descending*), misalnya jika ada data angka sebagai berikut :

3	5	2	7	9	0	1	4	6	8

maka jika diurutkan secara menaik (*ascending*) akan menjadi :

0	1	2	3	4	5	6	7	8	9

dan jika diurutkan secara menurun (*descending*) akan menjadi :

9	8	7	6	5	4	3	2	1	0

Jika yang diurutkan adalah sebuah rekaman (*record*) yang terdiri dari beberapa jenis data maka pengurutan biasanya dilakukan berdasarkan salah satu jenis data yang disimpan, misalnya jika sebuah rekaman (*record*) data mata kuliah mahasiswa dapat diurutkan berdasarkan nomor induk mahasiswa.

Pengurutan memiliki beberapa metode antara lain :

- Metode Penyisipan (*Insertion Sort*)
- Metode Seleksi
- Metode Gelembung (*Bubble Sort*)
- Metode QuickSort

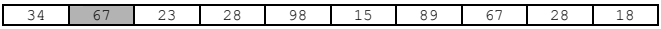
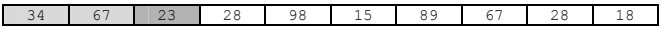
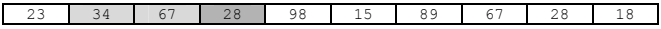
yang akan dibahas pada subbab-subbab selanjutnya. Pemilihan metode pengurutan bergantung pada kebutuhan dan kondisi karena setiap metode memiliki keunggulan dan kelemahan masing-masing. Selain metode di atas, masih banyak metode-metode lain yang tidak dibahas di dalam diktat ini (jika ingin tahu dapat melakukan eksplorasi).

3.1. Metode Penyisipan (Insertion Sort)

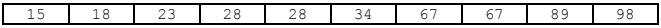
Metode penyisipan langsung (*straight insertion*) adalah metode pengurutan yang mengambil sebuah data sisip pada data yang diurutkan dan menggeser data yang lebih besar dari data sisip agar data sisip dapat ditempatkan pada tempat yang benar. Misalkan ada sebuah *array* yang berisi angka-angka sebagai berikut :

34	67	23	28	98	15	89	67	28	18

jika data di atas akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode penyisipan langsung (*straight insertion*) maka prosesnya adalah sebagai berikut :

Data Sisip	Hasil Pengurutan
67	 pada perulangan ke-1 data pada *array* indeks 2 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada sebelumnya, jika data sebelumnya tidak ada yang lebih besar dari data sisip maka tidak ada data yang harus digeser ke belakang
23	 pada perulangan ke-2 data pada *array* indeks 3 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat dua data yang lebih besar dibanding data sisip maka dua data itu harus digeser satu tempat dan data sisip dipindah ke tempat paling depan
28	 pada perulangan ke-3 data pada *array* indeks 4 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat dua data yang lebih besar dibanding data sisip maka dua data itu harus digeser satu tempat dan data sisip dipindah ke tempat sebelum dua data yang digeser
98	

Data Sisip	Hasil Pengurutan
	 pada perulangan ke-4 data pada *array* indeks 5 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada sebelumnya, jika data sebelumnya tidak ada yang lebih besar dari data sisip maka tidak ada data yang harus digeser ke belakang
15	 pada perulangan ke-5 data pada *array* indeks 6 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat lima data yang lebih besar dibanding data sisip maka lima data itu harus digeser satu tempat dan data sisip dipindah ke tempat paling depan
89	 pada perulangan ke-6 data pada *array* indeks 7 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat satu data yang lebih besar dibanding data sisip maka satu data itu harus digeser satu tempat dan data sisip dipindah ke tempat sebelum data yang digeser
67	 pada perulangan ke-7 data pada *array* indeks 8 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat dua data yang lebih besar dibanding data sisip maka dua data itu harus digeser satu tempat dan data sisip dipindah ke tempat sebelum dua data yang digeser
28	 pada perulangan ke-8 data pada *array* indeks 9 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat lima data yang lebih besar dibanding data sisip maka lima data itu harus digeser satu tempat dan data sisip dipindah ke tempat sebelum lima data yang digeser

Data Sisip	Hasil Pengurutan
18	 pada perulangan ke-9 data pada *array* indeks 10 dijadikan sebagai data sisip, kemudian dibandingkan dengan data yang ada di sebelumnya, terdapat delapan data yang lebih besar dibanding data sisip maka delapan data itu harus digeser satu tempat dan data sisip dipindah ke tempat sebelum delapan data yang digeser
Hasil Akhir	

Langkah-langkah di atas jika dituangkan dalam bahasa algoritmik adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan *array* yang berisi bilangan seperti contoh di atas	`tabInt : array [1…10] of integer`
	
	`tabInt <- {`
	`34, 67, 23, 28, 98, 15, 89, 67, 28, 18}`

mendeklarasikan bilangan counter sebagai counter perulangan	`i : integer`

mendeklarasikan perulangan yang memproses data sisip, didalamnya dideklarasikan bilangan counter untuk perulangan yang memproses pergeseran data	`data_sisip : integer`
	`j : integer`
	
	`for i<-2 to 10 do`
	
	`  data_sisip <- tabint`$_i$
	
	`  j <- i - 1`
	
	`  while (data_sisip < tabInt`$_j$`) and (j > 0) do`
	`  {jika data array lebih kecil dari data sisip maka data array digeser ke belakang}`
	
	`  tabInt`$_{j+1}$ `<- tabInt`$_j$
	`  j <- j - 1`
	
	`  {end while}`

Bahasa Manusia	Bahasa Algoritmik
	{*menempatkan data sisip pada array*} tabInt$_{j+1}$ <- data_sisip {*end for*}

Untuk mengubah pengurutan menaik (*ascending*) dengan metode penyisipan menjadi pengurutan menurun (*descending*) cukup dengan mengubah kode :

```
while (data_sisip < tabInt_j) and (j > 0) do
```

menjadi :

```
while (data_sisip > tabInt_j) and (j > 0) do
```

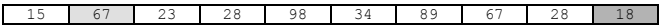
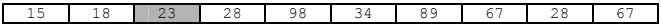
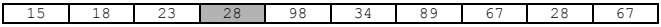
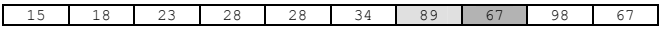
3.2. Metode Seleksi

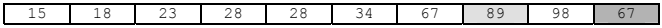
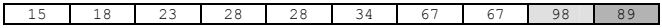
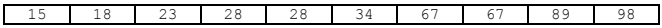
Metode seleksi adalah metode pengurutan yang mencari nilai terkecil atau terbesar bergantung pada pengurutan menaik atau menurun yang kemudian ditempatkan pada tempat paling depan, kemudian mencari lagi nilai terkecil atau terbesar kedua sepanjang jumlah elemen *array* dikurangi satu, setelah ketemu elemen kedua ditukar dengan nilai minimum, begitu seterusnya. Misalkan ada sebuah *array* yang berisi angka-angka sebagai berikut:

34	67	23	28	98	15	89	67	28	18

jika data di atas akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode seleksi maka prosesnya adalah sebagai berikut :

Nilai Min	Hasil Pengurutan
15	<table><tr><td>34</td><td>67</td><td>23</td><td>28</td><td>98</td><td>15</td><td>89</td><td>67</td><td>28</td><td>18</td></tr></table> pada perulangan ke-1 dicari nilai terkecil dari elemen *array* indeks 1 – 10 dan ditemukan nilai 15 sebagai nilai minimum, kemudian tempat nilai 15 ditukar dengan elemen paling depan

Nilai Min	Hasil Pengurutan
18	 pada perulangan ke-2 dicari nilai terkecil dari elemen *array* indeks 2 – 10 dan ditemukan nilai 18 sebagai nilai minimum, kemudian tempat nilai 18 ditukar dengan elemen indeks 2
23	 pada perulangan ke-3 dicari nilai terkecil dari elemen *array* indeks 3 – 10 dan ditemukan nilai 23 sebagai nilai minimum, elemen telah sesuai dengan tempatnya maka tidak perlu ditukar
28	 pada perulangan ke-4 dicari nilai terkecil dari elemen *array* indeks 4 – 10 dan ditemukan nilai 28 sebagai nilai minimum, elemen telah sesuai dengan tempatnya maka tidak perlu ditukar
28	 pada perulangan ke-5 dicari nilai terkecil dari elemen *array* indeks 5 – 10 dan ditemukan nilai 28 sebagai nilai minimum, kemudian tempat nilai 28 ditukar dengan elemen indeks 5
34	 pada perulangan ke-6 dicari nilai terkecil dari elemen *array* indeks 6 – 10 dan ditemukan nilai 34 sebagai nilai minimum, elemen telah sesuai dengan tempatnya maka tidak perlu ditukar
67	 pada perulangan ke-7 dicari nilai terkecil dari elemen *array* indeks 7 – 10 dan ditemukan nilai 67 sebagai nilai minimum, kemudian tempat nilai 67 ditukar dengan elemen indeks 7

Nilai Min	Hasil Pengurutan
67	 pada perulangan ke-8 dicari nilai terkecil dari elemen *array* indeks 8 – 10 dan ditemukan nilai 67 sebagai nilai minimum, kemudian tempat nilai 67 ditukar dengan elemen indeks 8
89	 pada perulangan ke-9 dicari nilai terkecil dari elemen *array* indeks 9 – 10 dan ditemukan nilai 89 sebagai nilai minimum, kemudian tempat nilai 89 ditukar dengan elemen indeks 9
Hasil Akhir	

Langkah-langkah di atas jika dituangkan dalam bahasa algoritmik adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan *array* yang berisi bilangan seperti contoh di atas	`tabInt : array [1…10] of integer`
	
	`tabInt <- {`
	`34, 67, 23, 28, 98, 15, 89, 67, 28, 18}`

mendeklarasikan bilangan counter sebagai counter perulangan	`i : integer`

mendeklarasikan perulangan yang memproses pertukaran nilai elemen dan pencarian nilai minimum	`minIndeks : integer`
	`j : integer`
	
	`for i<-1 to (10 - 1) do`
	
	`{inisialisasi indeks elemen minimum}`
	`minIndeks = tabInt`$_i$
	
	`{perulangan mencari nilai minimum sepanjang indeks i + 1 sampai jumlah elemen array}`
	
	`for j <- i + 1 to 10 do`
	`if tabInt`$_{minIndeks}$ `> tabInt`$_j$ `then`

Bahasa Manusia	Bahasa Algoritmik
	minIndeks <- j {*end if*} {*end for*} {*menukar posisi elemen*} integer temp <- $tabInt_i$ $tabInt_i$ <- $tabint_{minIndeks}$ $tabInt_{minIndeks}$ <- temp {*end for*}

Untuk mengubah pengurutan menaik (*ascending*) dengan metode seleksi menjadi pengurutan menurun (*descending*) cukup dengan mengubah kode :

if $tabInt_{minIndeks}$ > $tabInt_j$ then

menjadi :

if $tabInt_{minIndeks}$ < $tabInt_j$ then

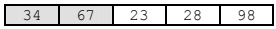
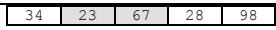
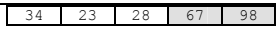
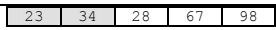
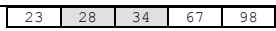
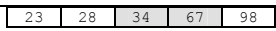
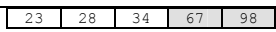
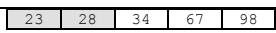
3.3. Metode Gelembung (Bubble Sort)

Metode gelembung atau *bubble sort* adalah metode pengurutan yang menukarkan dua buah elemen secara terus menerus sampai pengurutan selesai. Seringkali dikatakan bahwa metode ini kurang efisien namun sangat mudah dipahami. Misalkan ada sebuah *array* yang berisi angka-angka sebagai berikut:

34	67	23	28	98

jika data di atas akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode gelembung maka prosesnya adalah sebagai berikut :

Keterangan	Hasil Pengurutan

Keterangan	Hasil Pengurutan
perulangan ke-1 membandingkan elemen indeks 1 dengan indeks 2, karena elemen indeks 1 tidak lebih besar dari elemen indeks 2 maka tidak ada pertukaran	
perulangan ke-2 membandingkan elemen indeks 2 dengan indeks 3, karena elemen indeks 2 lebih besar dari elemen indeks 3 maka dilakukan pertukaran	
perulangan ke-3 membandingkan elemen indeks 3 dengan indeks 4, karena elemen indeks 3 lebih besar dari elemen indeks 4 maka dilakukan pertukaran	
perulangan ke-4 membandingkan elemen indeks 4 dengan indeks 5, karena elemen indeks 4 tidak lebih besar dari elemen indeks 5 maka tidak dilakukan pertukaran	
perulangan ke-5 membandingkan elemen indeks 1 dengan indeks 2, karena elemen indeks 1 tidak lebih besar dari elemen indeks 2 maka tidak ada pertukaran	
perulangan ke-6 membandingkan elemen indeks 2 dengan indeks 3, karena elemen indeks 2 tidak lebih besar dari elemen indeks 3 maka tidak dilakukan pertukaran	
perulangan ke-7 membandingkan elemen indeks 3 dengan indeks 4, karena elemen indeks 3 tidak lebih besar dari elemen indeks 4 maka tidak dilakukan pertukaran	
perulangan ke-8 membandingkan elemen indeks 4 dengan indeks 5, karena elemen indeks 4 tidak lebih besar dari elemen indeks 5 maka tidak dilakukan pertukaran	
perulangan ke-9 membandingkan elemen	

Keterangan	Hasil Pengurutan
indeks 1 dengan indeks 2, karena elemen indeks 1 tidak lebih besar dari elemen indeks 2 maka tidak ada pertukaran	
perulangan ke-10 membandingkan elemen indeks 2 dengan indeks 3, karena elemen indeks 2 tidak lebih besar dari elemen indeks 3 maka tidak dilakukan pertukaran	23 28 34 67 98
perulangan ke-11 membandingkan elemen indeks 3 dengan indeks 4, karena elemen indeks 3 tidak lebih besar dari elemen indeks 4 maka tidak dilakukan pertukaran	23 28 34 67 98
perulangan ke-12 membandingkan elemen indeks 4 dengan indeks 5, karena elemen indeks 4 tidak lebih besar dari elemen indeks 5 maka tidak dilakukan pertukaran	23 28 34 67 98
perulangan ke-13 membandingkan elemen indeks 1 dengan indeks 2, karena elemen indeks 1 tidak lebih besar dari elemen indeks 2 maka tidak ada pertukaran	23 28 34 67 98
perulangan ke-14 membandingkan elemen indeks 2 dengan indeks 3, karena elemen indeks 2 ltidak ebih besar dari elemen indeks 3 maka tidak dilakukan pertukaran	23 28 34 67 98
perulangan ke-15 membandingkan elemen indeks 3 dengan indeks 4, karena elemen indeks 3 tidak lebih besar dari elemen indeks 4 maka tidak dilakukan pertukaran	23 28 34 67 98
perulangan ke-16 membandingkan elemen indeks 4 dengan indeks 5, karena elemen indeks 4 tidak lebih besar dari elemen indeks 5 maka tidak dilakukan pertukaran	23 28 34 67 98

Langkah-langkah di atas jika dituangkan dalam bahasa algoritmik adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan *array* yang berisi bilangan seperti contoh di atas	`tabInt : array [1…10] of integer`
	
	`tabInt <- {`
	`34, 67, 23, 28, 98}`

mendeklarasikan bilangan counter sebagai counter perulangan, dan variabel temp untuk menyimpan nilai sementara, dan variabel penanda apakah masih terjadi pertukaran atau tidak	`i : integer`
	`temp : integer`
	`tukar : boolean`

mendeklarasikan perulangan yang memproses pertukaran nilai elemen	`repeat`
	
	`{inisialisasi nilai tukar sebelum ada pertukaran diset false}`
	`tukar <- false`
	
	`{pengulangan dan memeriksa apakah ada pertukaran}`
	`for i <- 1 to (5 - 1) do`
	
	`{jika ada nilai yang dipertukarkan}`
	`if` $tabInt_i$ `>` $tabInt_{i+1}$ `then`
	
	`{menukar posisi elemen}`
	$temp \leftarrow tabInt_i$
	$tabInt_i \leftarrow tabint_{i+1}$
	$tabInt_{i+1} \leftarrow temp$
	`tukar <- true`
	
	`{end if}`
	`{end for}`
	`until (tukar <> true)`

Untuk mengubah pengurutan menaik (*ascending*) dengan metode gelembung menjadi pengurutan menurun (*descending*) cukup dengan mengubah kode :

`if` $tabInt_i$ `>` $tabInt_{i+1}$ `then`

menjadi :

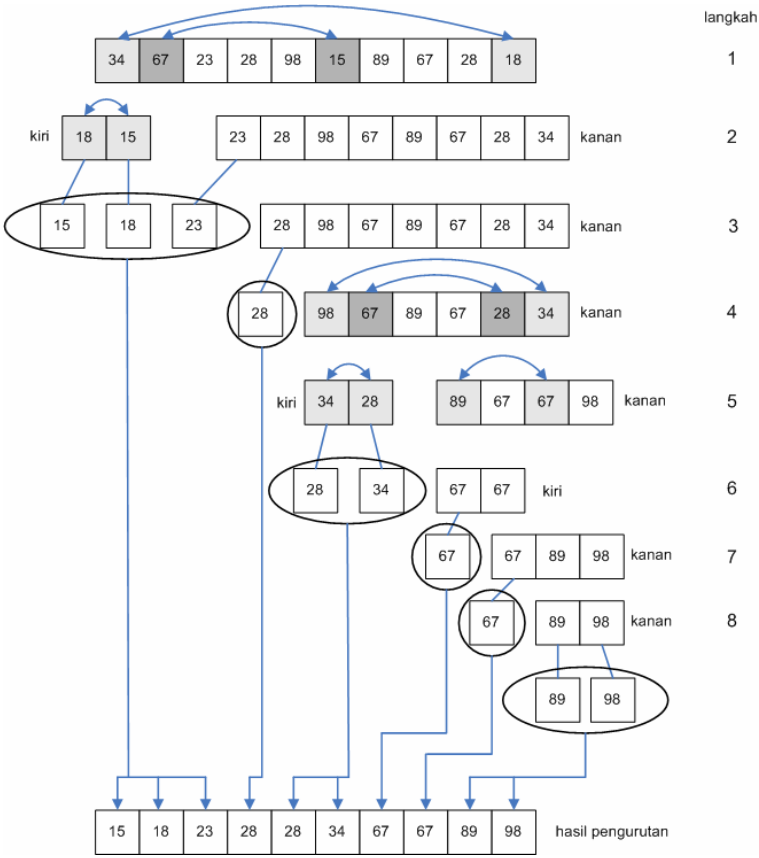
`if` $tabInt_i$ `<` $tabInt_{i+1}$ `then`

3.4. Metode QuickSort

Metode *quick sort* adalah metode pengurutan yang menjadikan sebuah tabel data yang akan diurutkan menjadi dua buah sub bagian yang ditelusuri dari kiri dan dari kanan. Misalkan ada sebuah *array* yang berisi angka-angka sebagai berikut:

34	67	23	28	98	15	89	67	28	18

jika data di atas akan diurutkan dengan urutan menaik (*ascending*) menggunakan metode *quick sort* maka prosesnya adalah sebagai berikut :



Langkah	Keterangan
1	• tabInt masih dianggap sebagai satu bagian yang ditelusuri dari kiri dan kanan • ketika nilai tabInt[1] sebagai penelusuran dari arah kiri lebih besar dari nilai tabInt[10] sebagai penelusuran dari arah kanan maka nilai tabInt[1] ditukar dengan tabInt[10] • penelusuran dari kanan berjalan ke kiri dan penelusuran dari kiri berjalan ke kanan • penelusuran berhenti saat nilai tabInt penelusuran dari

Langkah	Keterangan
	kiri lebih besar dari nilai tabInt penelusuran dari kanan yaitu tabInt[2] dan tabInt[6] kemudian nilai ditukar • penelusuran berjalan lagi sampai indeks penelusuran dari kiri sama dengan indeks penelusuran dari kanan
2	• tabInt dibagi menjadi sub bagian dimana sub bagian kiri adalah tabInt indeks 1 sampai nilai penelusuran kiri dan kanan bertemu, dan sub bagian kanan adalah tabInt indeks nilai penelusuran kiri dan kanan bertemu sampai indeks maksimum tabInt • langkah seperti langkah satu dilakukan pada sub bagian kiri dan sub bagian kanan • tabInt[1] ditukar dengan nilai tabInt[2] • pada sub bagian kanan, nilai tabInt[3] merupakan nilai terkecil dari sub bagian kanan, maka penelusuran akan bertemu pada nilai ini • nilai yang sudah terurut tidak diproses lagi yaitu tabInt[1], tabInt[2], dan tabInt[3]
3	• pada sub bagian kanan, nilai tabInt[4] merupakan nilai terkecil dari sub bagian kanan, maka penelusuran akan bertemu pada nilai ini • nilai tabInt[4] sudah terurut maka tidak diproses lagi sebagai sub bagian kanan
4	• langkah 1 dilakukan pada sub bagian kanan • nilai tabInt[5] ditukar dengan nilai tabInt[10] • nilai tabInt[6] ditukar dengan nilai tabInt[9]
5	• sub bagian kanan dibagi menjadi dua bagian menjadi sub bagian kiri dan sub bagian kanan • proses seperti langkah 1 dilakukan pada sub bagian kiri dan sub bagian kanan • nilai tabInt[5] ditukar dengan nilai tabInt[6] • nilai tabint[7] ditukar dengan nilai tabInt[9]
6	• sub bagian kanan dibagi menjadi dua menjadi sub bagian kiri dan sub bagian kanan • nilai tabInt[5] dan nilai tabInt[6] dianggap sudah terurut sehingga tidak diproses lagi • pada sub bagian kanan, nilai tabInt[7] merupakan nilai

Langkah	Keterangan
	terkecil dari sub bagian kanan, maka penelusuran akan bertemu pada nilai ini • nilai tabInt[7] sudah terurut maka tidak diproses lagi sebagai sub bagian kanan
7	• pada sub bagian kanan, nilai tabInt[8] merupakan nilai terkecil dari sub bagian kanan, maka penelusuran akan bertemu pada nilai ini • nilai tabInt[8] sudah terurut maka tidak diproses lagi sebagai sub bagian kanan
8	• sub bagian kanan memiliki dua buah nilai tabInt yang sudah terurut sehingga seluruh nilai tabInt sudah terurut menaik

Langkah-langkah di atas jika dituangkan dalam bahasa algoritmik adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan *array* yang berisi bilangan seperti contoh di atas	`tabInt : array [1…10] of integer`
	
	`tabInt <- {`
	`34, 67, 23, 28, 98, 15, 89, 67, 28, 18}`

mendeklarasikan prosedur quickSort
 	
i merupakan *counter* untuk penelusuran kiri dan j merupakan *counter* untuk penelusuran kanan
 	
perulangan proses pengurutan, dilakukan sampai i lebih besar dari j
 	
selama nilai penelusuran kiri lebih kecil dari nilai penelusuran kanan, penelusuran kiri maju
	`procedure quickSort(input : l : integer, r : integer)`
	
	`i : integer`
	`j : integer`
	
	`i <- l`
	`j <- r`
	
	`repeat`
	
	`while tabInt[i] < tabInt[l] do`
	`i <- i + 1`
	`{end while}`

Bahasa Manusia	Bahasa Algoritmik

selama nilai penelusuran kanan lebih besar dari nilai penelusuran kiri, penelusuran kanan maju	`while tabInt[j] > tabInt[1] do`
	`  j <- j - 1`
	`{end while}`

jika i lebih kecil j, maka nilai penelusuran kiri ditukar dengan nilai penelusuran kanan	`if i<j then`
	`  integer temp`
	`  temp <- tabInt[i]`
	`  tabInt[i] <- tabInt[j]`
	`  tabInt[j] <- temp`
	`  i <- i + 1`
	`  j <- j - 1`
	`{end if}`

jika sub bagian kiri mempunyai elemen, dilakukan pengurutan pada sub pohon kiri	`until(i>j)`
	
	`if l<j then`
	`  quickSort(l,j)`
	`{end if}`

jika sub bagian kanan mempunyai elemen, dilakukan pengurutan pada sub bagian kanan	`if i<r then`
	`  quickSort(i,r)`
	`{end if}`
	`{end procedure}`

mendeklarasikan prosedur menulis isi tabInt ke layar	`procedure tulis()`
	
	`  i : integer`
	`  for i<-1 to 10 do`
	
	`  write(tabInt[i])`
	
	`  {end for}`
	
	`{end procedure}`

mendeklarasikan program utama yang menampilkan isi tabInt sebelum dan sesudah diurutkan	`{program utama}`
	
	`tulis()`
	`quickSort(1,10)`
	`tulis()`
	
	`{end program utama}`

Untuk mengubah pengurutan menaik (*ascending*) dengan metode *quick sort* menjadi pengurutan menurun (*descending*) cukup dengan mengubah kode :

```
while tabInt[i] < tabInt[1] do
   i <- i + 1
{end while}

while tabInt[j] > tabInt[1] do
   j <- j - 1
{end while}
```

menjadi :

```
while tabInt[i] > tabInt[1] do
   i <- i + 1
{end while}

while tabInt[j] < tabInt[1] do
   j <- j - 1
{end while}
```

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan algoritma pengurutan yang telah dibahas untuk *array* dari tipe terstruktur titik, diurutkan berdasarkan nilai x atau nilai y dari titik.
2. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan algoritma pengurutan yang telah dibahas untuk *array* dari tipe terstruktur mahasiswa, diurutkan berdasarkan nilai nim dari mahasiswa.

4. Penggabungan Tabel (Merge)

4.1. Penggabungan Tabel Secara Tidak Terurut

Penggabungan tabel secara tidak terurut dapat langsung dilakukan dengan memasukkan isi tabel pertama ke dalam tabel ketiga diikuti dengan memasukkan isi tabel kedua ke dalam tabel ketiga. Berikut adalah algoritma penggabungan tabel secara tidak terurut:

Bahasa Manusia	Bahasa Algoritmik
Membuat 2 buah tabel yang akan diisi kemudian nantinya akan digabungkan	tabel1 : array [1..4] of integer tabel2 : array [1..4] of integer
Membuat tabel 3 untuk menyimpan hasil penggabungan tabel 1 dan 2	tabel3 : array [1..8] of integer
Membuat kotak penghitung untuk tabel 1 dan tabel 2	penghitung : integer
Melakukan pengulangan mengisi tabel 1 meminta masukan *user* /pemakai program untuk memasukkan elemen tabel 1	for penghitung <- 1 to 4 do input(tabel1$_{penghitung}$) {*end for*}
Melakukan pengulangan mengisi tabel 2 meminta masukan *user* /pemakai program untuk memasukkan elemen tabel 2	for penghitung <- 1 to 4 do input(tabel2$_{penghitung}$) {*end for*}

Bahasa Manusia	Bahasa Algoritmik
Melakukan pengulangan mengisi tabel 3 dengan isi dari tabel 1 elemen tabel 3 diisi dengan isi dari elemen tabel 1	for penghitung <- 1 to 4 do tabel3$_{penghitung}$ <- tabel1$_{penghitung}$ {end for}
Melakukan pengulangan mengisi tabel 3 dengan isi dari tabel 2 elemen tabel 3 diisi dengan isi dari elemen tabel 2	for penghitung <- 5 to 8 do tabel3$_{penghitung}$ <- tabel2$_{penghitung-4}$ {end for}

4.2. Penggabungan Tabel Secara Terurut

Penggabungan tabel secara terurut dilakukan dengan memeriksa elemen isi dari tabel 1 dan tabel 2, dibandingkan agar isi tabel 3 terurut. Berikut adalah algoritma penggabungan tabel secara terurut:

Bahasa Manusia	Bahasa Algoritmik
Membuat 2 buah tabel yang akan diisi kemudian nantinya akan digabungkan	tabel1 : array [1..4] of integer tabel2 : array [1..4] of integer
Membuat tabel 3 untuk menyimpan hasil penggabungan tabel 1 dan 2	tabel3 : array [1..8] of integer
Membuat kotak penghitung untuk tabel 1 dan tabel 2	penghitung1 : integer penghitung2 : integer penghitung3 : integer penghitung : integer
Melakukan pengulangan mengisi tabel 1 meminta masukan *user*	for penghitung1 <- 1 to 4 do input(tabel1$_{penghitung1}$)

Bahasa Manusia	Bahasa Algoritmik
/pemakai program untuk memasukkan elemen tabel 1	{*end for*}
Melakukan pengulangan mengisi tabel 2 meminta masukan *user* /pemakai program untuk memasukkan elemen tabel 2	for penghitung2 <- 1 to 4 do input(tabel2$_{penghitung2}$) {*end for*}

inisialisasi pengulangan
 	
Melakukan pengulangan mengisi tabel 3
 	
jika elemen tabel 1 lebih kecil dari tabel 2 maka
 	
masukkan elemen tabel 1 ke tabel 3
 	
iterasi penghitung tabel 1
 	
iterasi penghitung tabel 3
 	
jika elemen tabel 2 lebih kecil dari tabel 1 maka
 	
masukkan elemen tabel 2 ke tabel 3
 	
iterasi penghitung tabel 2
 	
iterasi penghitung tabel 3
	{*inisialisasi indeks awal setiap tabel*}
	penghitung1 <- 1
	penghitung2 <- 1
	penghitung3 <- 1
	
	while (penghitung1 <= 4) and (penghitung2 <=4) do
	
	if tabel1$_{penghitung1}$ < tabel2$_{penghitung2}$ then
	
	tabel3$_{penghitung3}$ <- tabel1$_{penghitung1}$
	
	penghitung1 <- penghitung1 + 1
	
	penghitung3 <- penghitung3 + 1
	
	{*end if*}
	
	else if tabel2$_{penghitung2}$ < tabel1$_{penghitung1}$ then
	
	tabel3$_{penghitung3}$ <- tabel2$_{penghitung2}$
	
	penghitung2 <- penghitung2 + 1

Bahasa Manusia	Bahasa Algoritmik
lainnya masukkan isi tabel 1 ke tabel 3 iterasi penghitung tabel 1 iterasi penghitung tabel 3 masukkan isi tabel 2 ke tabel 3 iterasi penghitung tabel 2 iterasi penghitung tabel 3	penghitung3 <- penghitung3 + 1 {*end if*} else $tabel3_{penghitung3}$ <- $tabel1_{penghitung1}$ penghitung1 <- penghitung1 + 1 penghitung3 <- penghitung3 + 1 $tabel3_{penghitung3}$ <- $tabel2_{penghitung2}$ penghitung2 <- penghitung2 + 1 penghitung3 <- penghitung3 + 1 {*end else*}

jika yang tersisa adalah tabel 1 maka masukkan isi sisa dari tabel 1 ke tabel 3	if penghitung1 < 4 then
	for penghitung <- penghitung1 to 4 do
	$tabel3_{penghitung3}$ <- $tabel1_{penghitung}$
	penghitung3 <- penghitung3 + 1
	{*end for*}
	{*end if*}

jika yang tersisa adalah tabel 2 maka masukkan isi sisa dari tabel 2 ke tabel 3	if penghitung2 < 4 then
	for penghitung <- penghitung2 to 4 do
	$tabel3_{penghitung3}$ <- $tabel2_{penghitung}$
	penghitung3 <- penghitung3 + 1
	{*end for*}

Bahasa Manusia	Bahasa Algoritmik
	{*end if*}

5. Pencarian

Pencarian atau *searching* suatu data pada sekumpulan data merupakan proses yang sangat penting dalam kehidupan nyata. Misalkan ada data sebagai berikut :

Nomor Induk	Nama	Nilai
13507701	Nana	64.75
13507702	Rudi	75.11
13507703	Dea	84.63
13507704	Ihsan	77.07
13507705	Tiara	66.70

kemudian dicari nilai dari Dea dimana data yang dipunyai adalah nomor induk Dea yaitu "13507703" maka setelah dilakukan proses pencarian, akan diketahui bahwa nilai Dea adalah 84,63.

Seperti halnya pengurutan, pencarian juga dapat dilakukan dengan beberapa metode pencarian seperti metode pencarian beruntun (*sequential search*) dan metode bagi dua (*binary search*) yang akan dibahas pada subbab-subbab berikut.

5.1. Pencarian Beruntun (Sequential Search)

Pencarian beruntun atau *sequential search* dapat dilakukan pada data yang belum terurut maupun yang sudah terurut. Pencarian beruntun dilakukan dengan melakukan penelusuran data satu persatu kemudian dicocokkan dengan data yang dicari, jika tidak sama maka penelusuran dilanjutkan, jika sama maka penelusuran dihentikan, berarti data telah ditemukan. Misalkan digunakan data seperti yang dicontohkan di atas maka langkah-langkah pencarian dengan pencarian beruntun adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan tipe nilaiMatKul untuk menyimpan struktur data yang akan dicari	type nilaiMatKul : <
	  nim : string,
	  nama : string,
	  nilai : real
	>

mendeklarasikan *array* yang nantinya akan diisi data	tabel : array [1…5] of nilaiMatKul

mendeklarasikan prosedur untuk mengisi tabel	procedure isi(input: i : integer; nim : string; nama : string; nilai : real)
	  $tabel_i$.nim <- nim
	  $tabel_i$.nama <- nama
	  $tabel_i$.nilai <- nilai
	{*end procedure*}

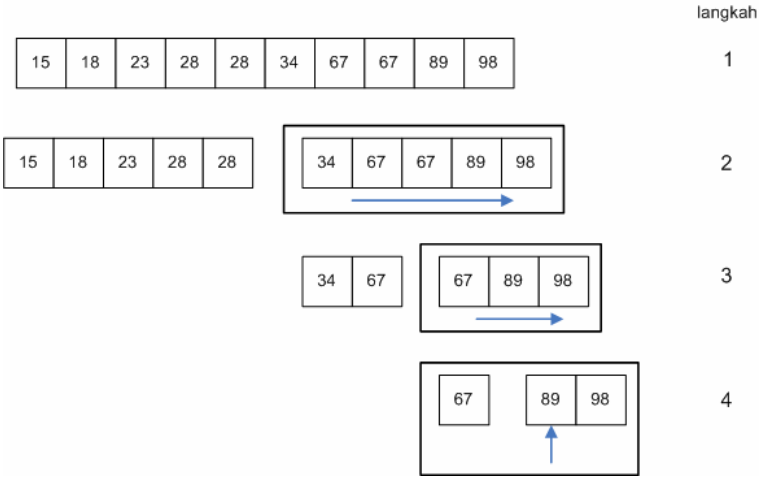
mendeklarasikan program utama yang mengisi tabel dengan data, dan melakukan pencarian dengan menggunakan pencarian beruntun	{*program utama*}
	
	  nim_cari : string
	  ketemu : boolean
	  i : integer
	
	  isi(1,"13507701","Nana",64.75)
	  isi(1,"13507702","Rudi",75.11)
	  isi(1,"13507703","Dea",84.63)
	  isi(1,"13507704","Ihsan",77.07)
	  isi(1,"13507705","Tiara",66.70)
	
	  nim_cari <- "13507703"
	  ketemu <- false
	  i <- 1
	
	  while (i <= 5) and (ketemu = false) do
	  if $tabel_i$ = nim_cari then
	  {*jika data ketemu*}
	  ketemu <- true
	  {*end if*}
	  else
	  i <- i + 1
	  {*end else*}
	  {*end while*}
	
	  if ketemu = true then
	  1. output("nim : ", $tabel_i$.nim)
	  output("nama : " $tabel_i$.nama)
	  output("nilai : ", $tabel_i$.nilai)
	  {*end if*}
	  else
	  output("tidak ditemukan")
	  {*end else*}
	{*end program utama*}

5.2. Pencarian Bagi Dua (Binary Search)

Pencarian bagi dua, atau pencarian biner, atau *binary search* hanya dapat dilakukan pada data yang sudah terurut. Misalkan ada sebuah *array* yang berisi angka-angka sebagai berikut:

34	67	23	28	98	15	89	67	28	18

jika yang dicari adalah angka 89 maka prosesnya adalah sebagai berikut :



Langkah	Keterangan
1	• tabel masih dianggap sebagai satu kesatuan dimana tabel berisi data yang telah terurut
2	• tabel dibagi menjadi dua, • kemudian dicek nilai paling kanan sub bagian kiri dan nilai paling kiri sub bagian kanan • jika nilai yang dicari lebih besar atau sama dengan nilai paling kiri sub bagian kanan, cari pada sub bagian kanan, jika nilai yang dicari lebih kecil atau

Langkah	Keterangan
	sama dengan nilai paling kanan sub bagian kiri maka cari pada sub bagian kiri, karena yang dicari adalah 89 maka dicari pada sub bagian kanan
3	• sub bagian kanan dibagi menjadi dua, • kemudian dicek nilai paling kanan sub bagian kiri dan nilai paling kiri sub bagian kanan • jika nilai yang dicari lebih besar atau sama dengan nilai paling kiri sub bagian kanan, cari pada sub bagian kanan, jika nilai yang dicari lebih kecil atau sama dengan nilai paling kanan sub bagian kiri maka cari pada sub bagian kiri, karena yang dicari adalah 89 maka dicari pada sub bagian kanan
4	• sub bagian kanan dibagi menjadi dua, • kemudian dicek nilai paling kanan sub bagian kiri dan nilai paling kiri sub bagian kanan • jika nilai yang dicari lebih besar atau sama dengan nilai paling kiri sub bagian kanan, cari pada sub bagian kanan, jika nilai yang dicari lebih kecil atau sama dengan nilai paling kanan sub bagian kiri maka cari pada sub bagian kiri, karena yang dicari adalah 89 maka sama dengan nilai paling kiri dari sub bagian kanan • nilai ditemukan

Langkah-langkah di atas jika dituangkan dalam bahasa algoritmik adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan *array* yang berisi bilangan seperti contoh di atas	`tabInt : array [1…10] of integer`
	
	`tabInt <- {`
	`34, 67, 23, 28, 98, 15, 89, 67, 28, 18}`

mendeklarasikan program utama yang melakukan pencarian dengan menggunakan pencarian bagi dua	`{program utama}`
	
	`i : integer`
	`j : integer`
	`bil_cari : integer`
	`k : integer`
	`ketemu : boolean`
	
	`i <- 1`

Bahasa Manusia	Bahasa Algoritmik

	j <- 10
	bil_cari <- 89
	ketemu <- false
	
	while (ketemu = false) and (i ≤ j) do
	k <- (i + j) div 2
	
	if $tabInt_k$ = bil_cari then
	ketemu <- true
	{*end if*}
	else
	if $tabInt_k$ > bil_cari then
	{*melakukan penelusuran sub bagian kiri*}
	j <- k - 1
	{*end if*}
	else
	{*melakukan penelusuran sub bagian kanan*}
	i <- k + 1
	{*end else*}
	{*end else*}
	{*end while*}
	
	if ketemu = true then
	output("ada pada tabel")
	{*end if*}
	else
	output("tidak ditemukan")
	{*end else*}
	
	{*end program utama*}

jika tabel yang digunakan terurut menurun, maka pencarian bagi dua dapat dilakukan dengan mengubah kode :

```
if tabInt_k > bil_cari then
{melakukan penelusuran sub bagian kiri}
   j <- k - 1
{end if}
 else
{melakukan penelusuran sub bagian kanan}
   i <- k + 1
{end else}
```

menjadi :

```
if tabInt_k < bil_cari then
{melakukan penelusuran sub bagian kiri}
   j <- k - 1
{end if}
 else
{melakukan penelusuran sub bagian kanan}
   i <- k + 1
{end else}
```

Soal Latihan:

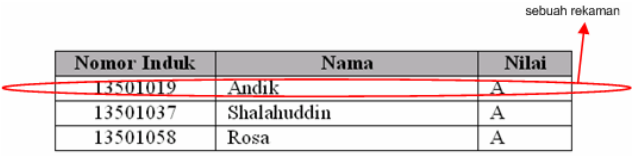
1. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan algoritma pencarian yang telah dibahas untuk *array* dari tipe terstruktur titik.
2. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan algoritma pencarian yang telah dibahas untuk *array* dari tipe terstruktur mahasiswa.

6. Arsip Beruntun (Sequential File)

6.1. Pendahuluan

6.1.1. Pengertian Rekaman (Record)

Rekaman atau *record* adalah data yang bertipe sama, misalnya ada beberapa data yang terdiri dari bagian-bagian tertentu seperti data nilai mahasiswa pada satu mata kuliah yang terdiri dari nomor induk, nama, dan nilai.



Nomor Induk	Nama	Nilai
13501019	Andik	A
13501037	Shalahuddin	A
13501058	Rosa	A

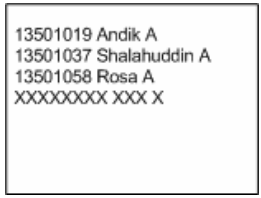
Gambar 3 Tabel Rekaman

Gambar 3 terdiri dari tiga buah rekaman yang memiliki kolom-kolom nomor induk, nama, dan nilai. Sebuah baris yang berisi data dari tiap kolom disebut dengan sebuah rekaman, namun dalam sebuah arsip beruntun biasanya terdiri dari banyak rekaman. Sebuah rekaman dapat terdiri dari beberapa kolom data tergantung dari kebutuhan.

6.1.2. Pengertian Arsip Beruntun (Sequential File)

Arsip beruntun atau *sequential file* adalah sebuah file yang berisi kumpulan rekaman dengan kolom-kolom data tertentu sesuai dengan kebutuhan, tapi dalam sebuah arsip beruntun, nama-nama kolom tidak ikut

disimpan di dalam file sehingga file hanya berisi kumpulan rekaman saja. Contoh isi sebuah arsip beruntun dapat dilihat pada gambar 4.



Gambar 4 Isi Arsip Beruntun

Pada sebuah file ada yang disebut dengan EOF (*end of file*) yang merupakan akhir dari sebuah file. Sebenarnya pada bahasa pemrograman Pascal dan C telah disediakan fungsi atau prosedur standar yang terkait dengan EOF ini sehingga dapat diketahui ketika pembacaan sebuah file bahwa sebuah file telah sampai pada akhir file atau belum. Namun dalam teori sebuah arsip beruntun perlu adanya sebuah rekaman sebagai tanda bahwa pembacaan rekaman pada arsip beruntun telah sampai pada rekaman akhir file yang biasanya berupa rekaman yang berisi sesuatu yang tidak mungkin merupakan data rekaman terserah pembuat program, misalnya XXXXXXXX XXX X untuk rekaman yang berisi nomor induk, nama, dan nilai dimana nilai nomor induk adalah XXXXXXXX, isi nama adalah XXX, dan isi nilai adalah X. Harus ada sebuah tanda pembatas antar nilai dan antar rekaman dimana pada contoh di atas pembatas antar nilai adalah spasi dan pembatas antar rekaman adalah *new line* . Jika pembacaan arsip beruntun telah sampai pada rekaman tanda akhir ini, maka pembacaan rekaman telah mencapai akhir file.

Rekaman penanda akhir *file* ini biasa disebut dengan *dummy* yang berarti elemen yang ditambahkan tapi sebenarnya bukan merupakan bagian dari data. *Dummy* biasa digunakan untuk memudahkan sebuah proses.

6.2. Operasi pada Arsip Beruntun

Arsip beruntun atau *sequential file* pada dasarnya adalah sebuah file yang dapat ditulisi untuk menyimpan data oleh karena itu maka operasi yang

digunakan pada arsip beruntun pada dasarnya adalah operasi yang dilakukan terhadap file yaitu menulisi file dan membaca file yang pada dasarnya sama dengan yang telah dijelaskan pada bab sebelumnya hanya saja membutuhkan beberapa perlakuan khusus atau menggunakan fungsi atau prosedur standar lain dari pustaka bahasa pemrograman yang lebih sesuai dengan operasi-operasi pada arsip beruntun.

6.2.1. Membuat Arsip Beruntun

Membuat arsip beruntun hanya membutuhkan beberapa data yang dibutuhkan dan menentukan rekaman *dummy* yang digunakan sebagai penanda akhir rekaman pada arsip beruntun. Hal-hal yang perlu diperhatikan untuk membuat sebuah arsip beruntun adalah sebagai berikut :

- **Menuliskan rekaman yang ingin ditulis**
  Sebuah arsip beruntun yang tidak kosong harus berisi satu atau lebih rekaman yang memiliki tipe nilai sama.
- **Menulis rekaman *dummy* sebagai akhir pembacaan rekaman**
  Jika sebuah arsip beruntun dianggap sebagai arsip kosong maka minimal ada satu buah rekaman *dummy* sebagai tanda akhir rekaman pada arsip beruntun, tapi jika arsip beruntun yang dibuat bukan arsip kosong maka rekaman *dummy* sebagai akhir rekaman harus ditulis pada file di akhir data rekaman.

Deklarasi dari sebuah arsip beruntun adalah sebagai berikut :

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
type nama_tipe : < nama_struktur1 : tipe_data, nama_struktur2 : tipe_data ..... > ..... nama_variabel_file : SeqFile of tipe_data nama_variabel_rekaman :	type nama_tipe = record nama_struktur1 : tipe_data; nama_struktur2 : tipe_data; ..... end; ..... var nama_variabel_file : text; nama_variabel_rekaman :	 typedef struct{ tipe_data nama_struktur1; tipe_data nama_struktur2; ..... }nama_tipe; ..... FILE *nama_variabel_file; nama_tipe

Bahasa Algoritmik	Bahasa Pascal	Bahasa C
nama_tipe	nama_tipe;	nama_variabel_rekaman;
	begin	

nama_variabel_file <- OPEN(nama_file , kode_ijin_akses)	assign(nama_variabel_file, nama_file);
	rewrite(nama_variabel_file);
	{atau
	reset(nama_variabel_file);}
		nama_variabel_file = fopen(nama_file, kode_ijin_akses);

WRITE(nama_variabel_file, string_yang_dituliskan)	write(nama_variabel_file, string_yang_dituliskan);	fprintf(nama__variabel_file, string_yang_dituliskan);
READ(nama_variabel_file, variabel_penyimpan_isi_file)	read(nama_variabel_file, variabel_penyimpan_isi_file);	fscanf(nama_variabel_file, string_pemesanan_tempat, &variabel_penyimpan_isi_file);
CLOSE(arsipMatKul)	close(nama_variabel_file); end.	fclose(nama_variabel_file);
	type	
type nilaiMatKul : < nim : string, nama : string, nilai : string >	nilaiMatKul = record nim : string; nama : string; nilai : string; end;	typedef struct{ char nim[10]; char nama[100]; char nilai[2]; } nilaiMatKul;
.....	.....	.....
	var	
arsipMatKul : SeqFile of nilaiMatKul	arsipMatKul : text;	FILE *arsipMatKul;
rekaman : nilaiMatKul	rekaman : nilaiMatKul;	nilaiMatKul rekaman;
	begin	

arsipMatKul <- OPEN("ArsipMatKul.dat", "w")	assign(arsipMatKul, 'ArsipMatKul.dat');
	rewrite(arsipMatKul);
	{atau
	reset(arsipMatKul);}
		arsipMatKul <- fopen("ArsipMatKul.dat", "w");

WRITE(arsipMatKul, rekaman.nim, " ", rekaman.nama, " ",	write(arsipMatKul, rekaman.nim + ' ' + rekaman.nama + ' ' +	fprintf(arsipMatKul, "%s %s %s\n", rekaman.nim, rekaman.nama,

Bahasa Algoritmik	Bahasa Pascal	Bahasa C

rekaman.nilai)
 	
READ(arsipMatKul,
rekaman.nim,
rekaman.nama,
rekaman.nilai)
 	
CLOSE(arsipMatKul)
	rekaman.nilai);
	
	read(arsipMatKul,
	rekaman.nim);
	
	close(arsipMatKul);
	end.
		rekaman.nilai);
		
		fscanf(arsipMatKul, "%s %s %s\n", &rekaman.nim, &rekaman.nama, &rekaman.nilai);
		
		fclose(arsipMatKul);

Pada bahasa pemrograman Pascal setelah menuliskan kode assign() setelah itu harus dituliskan kode

```
rewrite(nama_variabel_file);
```

jika *file* teks tidak dipastikan ada, sedangkan untuk membaca *file* teks maka *file* teks harus sudah ada sebelumnya maka setelah menuliskan kode assign() setelah itu harus dituliskan kode

```
reset(nama_variabel_file);
```

Berikut adalah algoritma arsip beruntun yang berisi rekaman nilai mata kuliah yang terdiri dari nomor induk, nama, dan nilai mata kuliah:

Bahasa Manusia	Bahasa Algoritmik

Mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file dan membuka file dengan kode akses penulisan	type nilaiMatKul : <
	  nim : string,
	  nama : string,
	  nilai : string
	>
	
	rekaman : nilaiMatKul
	
	arsipMatKul : SeqFile of nilaiMatKul
	
	arsipMatKul <- OPEN("ArsipMatKul.dat", "w")

Menuliskan ke layar untuk menuliskan rekaman satu per satu	output("Tuliskan rekaman satu per satu : ")
membaca nilai rekaman pertama sebagai inisialisasi dari perulangan memasukkan rekaman	output("masukkan nim : ") input(rekaman.nim)
membuat proses perulangan untuk memasukkan rekaman satu per satu	while rekaman.nim <> "XXXXXXXX" do {*proses*}

Bahasa Manusia	Bahasa Algoritmik
dan menuliskannya ke file sampai ditemui rekaman *dummy*	output("masukkan nama : ") input(rekaman.nama) output("masukkan nilai : ") input(rekaman.nilai) {*menuliskan ke file*} WRITE(arsipMatKul, rekaman.nim, " ", rekaman.nama, " ", rekaman.nilai) {*iterasi*} output("masukkan nim : ") input(rekaman.nim) {*end while*}
Menuliskan rekaman *dummy*	WRITE(arsipMatKul, "XXXXXXXX XXX X")
Menutup file	CLOSE(arsipMatKul)

6.2.2. Membaca Arsip Beruntun

Kasus-kasus yang harus diperhatikan pada saat pembacaan sebuah arsip beruntun adalah sebagai berikut :

- **Arsip kosong**
  Ada kemungkinan sebuah arsip beruntun tidak berisi rekaman data dan hanya berisi rekaman *dummy* sebagai tanda akhir maka arsip beruntun dianggap sebagai arsip kosong. Arsip seperti ini tidak memerlukan pemrosesan pembacaan karena arsip kosong. Contoh isi sebuah arsip kosong dapat dilihat pada gambar 5.

Gambar 5 Arsip Kosong

- **Arsip berisi rekaman**
  Diperlukan proses untuk membaca arsip beruntun sampai rekaman yang terakhir.

6.2.2.1. Membaca Arsip Beruntun tanpa Proses Pencarian

Berikut adalah sebuah algoritma untuk membaca sebuah arsip beruntun yang berisi rekaman nilai mata kuliah yang terdiri dari nomor induk, nama, dan nilai mata kuliah yang telah dibuat:

Bahasa Manusia	Bahasa Algoritmik

Mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file dan membuka file dengan kode akses penulisan	`type nilaiMatKul : <`
	`  nim : string,`
	`  nama : string,`
	`  nilai : string`
	`>`
	
	`rekaman : nilaiMatKul`
	
	`arsipMatKul : SeqFile of nilaiMatKul`
	
	`arsipMatKul <- OPEN("ArsipMatKul.dat", "r")`

Menuliskan ke layar untuk menampilkan rekaman satu per satu	`output("isi arsip beruntun adalah : ")`
Membaca nilai rekaman pertama dari file	`READ(arsipMatKul, rekaman.nim, rekaman.nama, rekaman.nilai)`

Membuat penanganan jika arsip beruntun merupakan arsip kosong	`if rekaman.nim = "XXXXXXXX" then`
	`  output("arsip kosong")`
	`{end if}`

Membuat proses perulangan untuk membaca rekaman satu per satu dan menampilkannya ke layar sampai ditemui rekaman *dummy*	`else`
	`repeat`
	
	`  {proses}`
	`  output("nim : ", rekaman.nim)`
	`  output("nama : ", rekaman.nama)`
	`  output("nilai : ", rekaman.nilai)`
	`  output("--------------------------")`
	
	`  {iterasi}`
	`  READ(arsipMatKul, rekaman.nim, rekaman.nama, rekaman.nilai)`
	
	`until rekaman.nim = "XXXXXXXX"`
	`{end else}`

Menutup file	`CLOSE(arsipMatKul)`

6.2.2.2. Membaca Arsip Beruntun dengan Pencarian

Berikut adalah sebuah algoritma untuk membaca sebuah arsip beruntun yang berisi rekaman nilai mata kuliah yang terdiri dari nomor induk, nama, dan nilai mata kuliah yang telah dibuat sebelumnya dengan pencarian berdasarkan nomor induk:

Bahasa Manusia	Bahasa Algoritmik

Mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file dan membuka file dengan kode akses penulisan	`type nilaiMatKul : <`
	`  nim : string,`
	`  nama : string,`
	`  nilai : string`
	`>`
	
	`rekaman : nilaiMatKul`
	
	`arsipMatKul : SeqFile of nilaiMatKul`
	
	`arsipMatKul <- OPEN("ArsipMatKul.dat", "r")`

Mendeklarasikan variabel string yang menyimpan kata yang dicari dan menuliskan ke layar untuk memasukkan kata kunci nama yang akan dicari	`kata_cari : string`
	
	`output("masukkan nomor induk yang dicari : ")`
	`input(kata_cari)`

Membaca nilai rekaman pertama dari file	`READ(arsipMatKul, rekaman.nim, rekaman.nama, rekaman.nilai)`

Membuat penanganan jika arsip beruntun merupakan arsip kosong	`if rekaman.nim = "XXXXXXXX" then`
	`  output("arsip kosong");`
	`{end if}`

Membuat penanganan jika masukan adalah nilai nomor induk pada rekaman *dummy* yaitu "XXXXXXXX", karena rekaman ini sebenarnya hanya sebagai penambah maka rekaman dengan nomor induk "XXXXXXXX" dianggap tidak ada	`else`
	` if kata_kunci = "XXXXXXXX" then`
	`  output("tidak ditemukan")`
	` {end if}`

Membuat proses perulangan untuk membaca rekaman satu per satu dan mencari nama yang dicari sampai ditemui rekaman *dummy* atau nama yang dicari telah	`else`
	`while (rekaman.nim <> "XXXXXXXX") and (rekaman.nim <> kata_cari) do`
	`  {iterasi}`
	`  READ(arsipMatKul, rekaman.nim, rekaman.nama, rekaman.nilai)`
	`{end while}`

Bahasa Manusia	Bahasa Algoritmik

ditemukan	`if rekaman.nim = kata_cari then`
	`  output("hasil pencarian : ")`
	`  output("nim : ", rekaman.nim)`
	`  output("nama : ", rekaman.nama)`
	`  output("nilai : ", rekaman.nilai)`
	`  output("--------------------------")`
	`{end if}`
	`else`
	`  output("tidak ditemukan")`
	`{end else}`
	
	`{end else}`

Menutup file	`CLOSE(arsipMatKul)`

6.2.3. Menyalin Arsip Beruntun

Pada subbab ini juga akan dibahas bagaimana menyalin sebuah arsip beruntun mulai dari rekaman pertama sampai rekaman *dummy* ke arsip beruntun lainnya. Rekaman yang disalin dapat semua rekaman atau rekaman tertentu saja dengan pemrosesan tertentu, dan jika memenuhi syarat, baru rekaman akan ditulis ke file lainnya. Langkah yang harus dilakukan jika tidak semua rekaman pada file pertama disalin ke file kedua maka pembacaan dilakukan pada file pertama per rekaman, setiap kali rekaman dibaca dicek apakah rekaman memenuhi syarat untuk ditulis pada file kedua, jika tidak maka baca rekaman berikutnya pada file pertama, dicek kembali apakah memenuhi syarat untuk ditulis pada file kedua, jika memenuhi syarat maka rekaman ditulis pada file kedua.

Berikut adalah sebuah algoritma untuk menyalin sebuah arsip beruntun ke arsip beruntun lainnya dimana rekaman terdiri dari nomor induk, nama, dan nilai seperti yang telah dibuat:

Bahasa Manusia	Bahasa Algoritmik

mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file pertama dan kedua dan membuka file dengan kode akses pembacaan untuk file pertama dan kode akses penulisan untuk file kedua	`type nilaiMatKul : <`
	`  nim : string,`
	`  nama : string,`
	`  nilai : string`
	`>`
	
	`rekaman : nilaiMatKul`
	
	`arsipMatKul1 : SeqFile of nilaiMatKul`
	
	`arsipMatkul2 : SeqFile of nilaiMatKul`
	
	`arsipMatKul1 <- OPEN("ArsipMatKul.dat", "r")`

Bahasa Manusia	Bahasa Algoritmik
	`arsipMatKul2 <- OPEN("ArsipMatKul.bak", "w")`

membaca nilai rekaman pertama dari file pertama	`output("mulai proses menyalin file")`
	
	`READ(arsipMatKul1, rekaman.nim, rekaman.nama, rekaman.nilai)`

membuat penanganan jika arsip beruntun merupakan arsip kosong, maka rekaman *dummy* langsung dituliskan pada file kedua	`if rekaman.nim = "XXXXXXXX" then`
	`  WRITE(arsipMatKul2, "XXXXXXXX XXX X")`
	`{end if}`

membuat proses perulangan untuk membaca rekaman satu per satu dari file pertama dan menulisnya pada file kedua sampai ditemui rekaman *dummy* kemudian menuliskan rekaman *dummy* ke file kedua	`else`
	`  while rekaman.nim <> "XXXXXXXX" do`
	`    WRITE(arsipMatKul2, rekaman.nim, " ", rekaman.nama, " ", rekaman.nilai, "\n")`
	`    READ(arsipMatKul1, rekaman.nim, rekaman.nama, rekaman.nilai)`
	`  {end while}`
	
	`  WRITE(arsipMatKul2, "XXXXXXXX XXX X")`
	`{end else}`

menutup file	`output("proses menyalin file selesai")`
	
	`CLOSE(arsipMatKul1)`
	`CLOSE(arsipMatKul2)`

Metode untuk menyalin arsip beruntun di atas juga dapat digunakan untuk melakukan pengubahan rekaman (*update*) dengan cara menyalin isi file ke file sementara dan menulisnya ulang pada file sebenarnya dengan rekaman yang telah diproses untuk keperluan *update*.

6.2.4. Penggabungan Arsip Beruntun (merging)

Penggabungan arsip beruntun atau *merging* merupakan operasi pada arsip beruntun dimana rekaman pada file pertama digabungkan dengan

rekaman pada arsip kedua dan disimpan dalam sebuah arsip beruntun lainnya. Misalkan ada dua buah arsip beruntun berikut :

ArsipMatKul1 :

Nomor Induk	Nama	Nilai
13501019	Andik	A
13501037	Shalahuddin	A
13501058	Rosa	A
XXXXXXXX	XXX	X

ArsipMatKul2 :

Nomor Induk	Nama	Nilai
13501004	Hervin	A
13501057	Ayub	A
13501069	Farah	A
13501084	Eni	A
XXXXXXXX	XXX	X

maka kedua arsip beruntun di atas jika digabungkan secara penyambungan (*concate*) akan menjadi :

ArsipMatKul3 :

Nomor Induk	Nama	Nilai
13501019	Andik	A
13501037	Shalahuddin	A
13501058	Rosa	A
13501004	Hervin	A
13501057	Ayub	A
13501069	Farah	A
13501084	Eni	A
XXXXXXXX	XXX	X

dan jika digabungkan dengan terurut akan menjadi :

ArsipMatKul4 :

Nomor Induk	Nama	Nilai
13501004	Hervin	A
13501019	Andik	A

Nomor Induk	Nama	Nilai
13501037	Shalahuddin	A
13501057	Ayub	A
13501058	Rosa	A
13501069	Farah	A
13501084	Eni	A
XXXXXXXX	XXX	X

Mengenai algoritma dan latihan program penyambungan akan dibahas pada subbab berikutnya.

6.2.4.1. Penggabungan Arsip Beruntun dengan Penyambungan (concate)

Sebuah program untuk menggabungkan dua buah arsip beruntun dengan penyambungan dapat dibuat sebagai latihan dimana kumpulan rekaman yang ada pada arsip beruntun kedua akan ditulis setelah kumpulan rekaman pada arsip beruntun yang pertama. Rekaman yang digunakan adalah rekaman yang berisi nomor induk, nama, dan nilai seperti yang telah dicontohkan di atas. Kondisi awal file ArsipMatKul1.dat telah ada dan berisi :

```
13501019 Andik A
13501037 Shalahuddin A
13501058 Rosa A
XXXXXXXX XXX X
```

dan file ArsipMatKul2.dat juga telah berisi rekaman :

```
13501004 Hervin A
13501057 Ayub A
13501069 Farah A
13501084 Eni A
XXXXXXXX XXX X
```

Langkah-langkah yang harus dilakukan adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

Mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file pertama,	`type nilaiMatKul : <`
	`  nim : string,`
	`  nama : string,`
	`  nilai : string`
	`>`

Bahasa Manusia	Bahasa Algoritmik
kedua, dan ketiga serta membuka file dengan kode akses pembacaan untuk file pertama dan kedua, kode akses penulisan untuk file ketiga	rekaman : nilaiMatKul arsipMatKul1 : SeqFile of nilaiMatKul arsipMatkul2 : SeqFile of nilaiMatKul arsipMatkul3 : SeqFile of nilaiMatKul arsipMatKul1 <- OPEN("ArsipMatKul1.dat", "r") arsipMatKul2 <- OPEN("ArsipMatKul2.dat", "r") arsipMatKul3 <- OPEN("ArsipMatKul3.dat", "w")
Membaca nilai rekaman pertama dari file pertama	output("mulai proses penggabungan file") READ(arsipMatKul1, rekaman.nim, rekaman.nama, rekaman.nilai)

Membuat proses perulangan untuk membaca rekaman satu per satu dari file pertama dan menulisnya pada file ketiga sampai ditemui rekaman *dummy*	while rekaman.nim <> "XXXXXXXX" do
	    WRITE(arsipMatKul3, rekaman.nim, " ", rekaman.nama, " ", rekaman.nilai, "\n")
	    READ(arsipMatKul1, rekaman.nim, rekaman.nama, rekaman.nilai)
	{*end while*}

Membaca nilai rekaman pertama dari file kedua	READ(arsipMatKul2, rekaman.nim, rekaman.nama, rekaman.nilai)

Membuat proses perulangan untuk membaca rekaman satu per satu dari file kedua dan menulisnya pada file ketiga sampai ditemui rekaman *dummy*	while rekaman.nim <> "XXXXXXXX" do
	    WRITE(arsipMatKul3, rekaman.nim, " ", rekaman.nama, " ", rekaman.nilai, "\n")
	    READ(arsipMatKul2, rekaman.nim, rekaman.nama, rekaman.nilai)
	{*end while*}

Menulis rekaman *dummy* pada file ketiga dan menutup file semua file	WRITE(arsipMatKul3, "XXXXXXXX XXX X") output("proses penggabungan file selesai") CLOSE(arsipMatKul1) CLOSE(arsipMatKul2) CLOSE(arsipMatKul3)

6.2.4.2. Penggabungan Arsip Beruntun dengan Terurut

Sebuah program untuk menggabungkan dua buah arsip beruntun dengan terurut dapat dibuat sebagai latihan dimana kumpulan rekaman yang ada pada arsip beruntun pertama dan kedua akan ditulis secara terurut nomor induk pada arsip beruntun yang baru. Rekaman yang digunakan adalah rekaman yang berisi nomor induk, nama, dan nilai seperti yang telah dicontohkan di atas. Kondisi awal file ArsipMatKul1.dat telah ada dan berisi :

```
13501019 Andik A
13501037 Shalahuddin A
13501058 Rosa A
XXXXXXXX XXX X
```

dan file ArsipMatKul2.dat juga telah berisi rekaman :

```
13501004 Hervin A
13501057 Ayub A
13501069 Farah A
13501084 Eni A
XXXXXXXX XXX X
```

Langkah-langkah yang harus dilakukan adalah sebagai berikut :

Bahasa Manusia	Bahasa Algoritmik

Mendeklarasikan struktur tipe rekaman, variabel rekaman, variabel yang akan mengacu pada file pertama, kedua, dan ketiga serta membuka file dengan kode akses pembacaan untuk file pertama dan kedua, kode akses penulisan untuk file ketiga	`type nilaiMatKul : <`
	`  nim : string,`
	`  nama : string,`
	`  nilai : string`
	`>`
	
	`rekaman1 : nilaiMatKul`
	
	`rekaman2 : nilaiMatKul`
	
	`arsipMatKul1 : SeqFile of nilaiMatKul`
	
	`arsipMatkul2 : SeqFile of nilaiMatKul`
	
	`arsipMatkul3 : SeqFile of nilaiMatKul`
	
	`arsipMatKul1 <-`
	`OPEN("ArsipMatKul1.dat", "r")`
	
	`arsipMatKul2 <-`
	`OPEN("ArsipMatKul2.dat", "r")`
	
	`arsipMatKul3 <-`

Bahasa Manusia	Bahasa Algoritmik
	OPEN("ArsipMatKul4.dat", "w")
Membaca nilai rekaman pertama dari file pertama dan file kedua	output("mulai proses penggabungan file") READ(arsipMatKul1, rekaman1.nim, rekaman1.nama, rekaman1.nilai) READ(arsipMatKul2, rekaman2.nim, rekaman2.nama, rekaman2.nilai)

Membuat proses perulangan untuk membaca rekaman satu per satu dari file pertama dan file kedua, membandingkannya dan menulis rekaman secara terurut pada file ketiga sampai ditemui rekaman *dummy* pada file pertama atau file kedua	while rekaman1.nim <> "XXXXXXXX" and rekaman2.nim <> "XXXXXXXX" do
	  if rekaman1.nim ≤ rekaman2.nim
	  WRITE(arsipMatKul3, rekaman1.nim, " ", rekaman1.nama, " ", rekaman1.nilai, "\n")
	  READ(arsipMatKul1, rekaman1.nim, rekaman1.nama, rekaman1.nilai)
	  {*end if*}
	  else
	  WRITE(arsipMatKul3, rekaman2.nim, " ", rekaman2.nama, " ", rekaman2.nilai, "\n")
	  READ(arsipMatKul2, rekaman2.nim, rekaman2.nama, rekaman2.nilai)
	  {*end else*}
	{*end while*}

Membuat proses perulangan untuk membaca rekaman satu per satu dari file pertama jika file pertama masih tersisa rekaman yang belum terbaca dan menulisnya pada file ketiga sampai ditemui rekaman *dummy* pada file pertama	while rekaman1.nim <> "XXXXXXXX" do
	  WRITE(arsipMatKul3, rekaman1.nim, " ", rekaman1.nama, " ", rekaman1.nilai, "\n")
	  READ(arsipMatKul1, rekaman1.nim, rekaman1.nama, rekaman1.nilai)
	{*end while*}

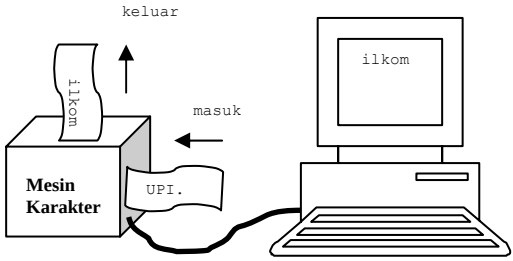
Membuat proses perulangan untuk membaca rekaman satu per satu dari file kedua jika file kedua masih tersisa rekaman yang belum terbaca dan menulisnya pada file ketiga sampai ditemui rekaman *dummy* pada file kedua	while rekaman2.nim <> "XXXXXXXX" do
	  WRITE(arsipMatKul3, rekaman2.nim, " ", rekaman2.nama, " ", rekaman2.nilai, "\n")
	  READ(arsipMatKul2, rekaman2.nim, rekaman2.nama, rekaman2.nilai)
	{*end while*}

Menulis rekaman *dummy* pada file ketiga dan menutup file semua file	WRITE(arsipMatKul3, "XXXXXXXX XXX X") output("proses penggabungan file selesai") CLOSE(arsipMatKul1) CLOSE(arsipMatKul2) CLOSE(arsipMatKul3)

7. Mesin Abstrak

Sebutan mesin abstrak sebenarnya adalah sebuah program yang dibuat terstruktur layaknya komponen-komponen pada mesin secara riil. Misalkan dibuat sebuah mesin abstrak untuk membuat gambar, maka akan dibuat kumpulan prosedur dan fungsi yang melakukan pemrosesan gambar agar program menggambar menggunakan mesin abstrak gambar dapat dijalankan. Prosedur-prosedur yang dibuat misalnya prosedur untuk menggambar titik, prosedur untuk menggambar garis, prosedur untuk mengeset warna titik atau garis dan lain sebagainya. Fungsi-fungsi yang dibuat misalnya mengembalikan titik terakhir pena digerakkan dan lain sebagainya. Mesin abstrak untuk menggambar atau sering disebut mesin gambar misalnya adalah seperti program paint di sistem operasi windows.

Contoh mesin abstrak yang tidak terlalu rumit salah satunya adalah mesin karakter. Mesin karakter digunakan untuk memanipulasi *string* atau memproses isi sebuah *file*. Proses-proses itu antara lain misalnya membaca teks dari *file*, melakukan penggantian isi *file*, memproses setiap karakter dalam teks satu per satu, dan lain sebagainya. Gambaran mesin karakter secara logika dapat dilihat pada gambar 6.



Gambar 6 Mesin Karakter

7.1. Mesin Karakter

7.1.1. Mesin Karakter untuk Pemrosesan Per Karakter

Dengan membuat mesin karakter, maka dapat diaplikasikan berbagai kasus untuk manipulasi karakter pada teks. Komponen-komponen utama pada mesin karakter antara lain:

Komponen	Keterangan
START	merupakan komponen yang bertugas melakukan inisialisasi pada semua atribut mesin karakter prosedur start biasanya diawali dengan membuka *file* teks yang berisi teks jika menggunakan *file* teks, atau berisi inisialisasi jika teks berasal dari sebuah *string*
CC	merupakan komponen penunjuk karakter yang saat ini sedang dibaca (*current character*) CC merupakan variabel yang menyimpan karakter yang sedang dibaca saat ini
INC	merupakan komponen yang bertugas memajukan CC satu karakter ke depan prosedur inc biasanya berisi memajukan pembacaan karakter maju satu karakter (diubah isi CC-nya menjadi karakter di depannya)
ADV	merupakan komponen yang bertugas maju ke karakter berikutnya tapi jika bertemu *blank* atau spasi maka akan maju lagi satu karakter, komponen ini menggunakan komponen INC prosedur adv biasanya berisi pembacaan karakter, jika bertemu karakter *blank* atau spasi maka akan maju satu kali lagi hingga ketemu karakter bukan *blank*

Komponen	Keterangan
EOP	merupakan komponen yang menjadi penanda akhir proses (misalnya karakter titik) atau akhir pembacaan teks atau akhir *file* fungsi eop biasanya berisi pemeriksaan pembacaan apakah bertemu karakter penanda akhir teks atau akhir *file* (jika menggunakan *file*)

Komponen-komponen fungsi yang sering digunakan antara lain:

Fungsi	Keterangan
Baca dan Tampilkan Karakter	Membaca dan menampilkan karakter satu per satu
Hitung Karakter	Menghitung karakter satu per satu

Semua komponen mesin karakter dapat digunakan untuk aplikasi yang lebih besar.

Berikut adalah contoh mesin karakter yang menggunakan teks berupa string (*array* karakter) (dalam bahasa algoritmik):

```
CCIndeks : integer
CC : char


procedure START(teks : string)
{inisialisasi awal pemrosesan mesin karakter}

   CCIndeks <- 1
   CC <- teks_CCIndeks

{end procedure}

procedure INC(teks : string)
{maju satu karakter}

   CCIndeks <- CCIndeks + 1
   CC <- teks_CCIndeks

{end procedure}

procedure ADV(teks : string)
{maju satu karakter, jika bertemu blank maka maju lagi satu karakter}

   repeat

      INC(teks)
```

```
   until (CC <> ' ')

{end procedure}

function EOP(teks : string) : boolean
{mengembalikan status apakah sudah bertemu akhir teks atau akhir array}

   if CC = '.' or CCIndeks > length(teks) then

        -> true
   {end if}
   else
        -> false
   {end else}

{end function}
```

Berikut adalah algoritma untuk proses pembacaan dan menampilkan karakter satu per satu menggunakan mesin karakter di atas (dalam bahasa algoritmik):

```
{program utama}

   teks : string

   input(teks)

   START(teks)

   while (not EOP(teks)) do

      output(CC)
      INC(teks)

   {end while}

{end program utama}
```

Berikut adalah algoritma untuk proses penghitungan karakter tanpa menghitung *blank* atau spasi menggunakan mesin karakter di atas (dalam bahasa algoritmik):

```
{program utama}

   teks : string
   jumlah_huruf : integer

   input(teks)

   START(teks)

   jumlah_huruf <- 0

   while (not EOP(teks)) do
```

```
    jumlah_huruf <- jumlah_huruf + 1
    ADV(teks)

  {end while}

  output(jumlah_huruf)

{end program utama}
```

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan mesin karakter untuk menghitung banyaknya huruf 'A' atau 'a' pada sebuah teks.
2. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan mesin karakter untuk menghitung banyaknya huruf hidup pada sebuah teks.
3. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan mesin karakter untuk menghitung banyaknya huruf mati pada sebuah teks.
4. Buatlah mesik karakter dengan teks dibaca dari *file* teks.
5. Kerjakan soal nomor 1 sampai 3 dengan menggunakan *file* teks.

7.1.2. Mesin Karakter untuk Pemrosesan Per Kata (Mesin Kata)

Dengan membuat mesin kata, maka dapat diaplikasikan berbagai kasus untuk manipulasi kata pada teks. Mesin kata menggunakan mesin karakter untuk memproses per karakter dan menyusun kata. Komponen-komponen utama pada mesin kata antara lain:

Komponen	Keterangan
STARTKATA	merupakan komponen yang bertugas melakukan inisialisasi pada semua atribut mesin kata prosedur startkata biasanya diawali dengan membuka *file* teks yang berisi teks jika menggunakan *file* teks, atau berisi inisialisasi jika teks berasal dari sebuah *string*
RESETKATA	merupakan komponen yang bertugas

	mengosongkan CCKATA
CCKATA	merupakan komponen penunjuk kata yang saat ini sedang dibaca (*current word*) CCKATA merupakan variabel yang menyimpan kata yang sedang dibaca saat ini
INCKATA	merupakan komponen yang bertugas memajukan CCKATA satu kata ke depan prosedur inckata biasanya berisi memajukan pembacaan kata maju satu kata (diubah isi CCKATA-nya menjadi kata di depannya)
EOPKATA	merupakan komponen yang menjadi penanda akhir proses (misalnya karakter titik) atau akhir pembacaan teks atau akhir *file* fungsi eopkata biasanya berisi pemeriksaan pembacaan apakah bertemu karakter penanda akhir teks atau akhir *file* (jika menggunakan *file*)

Komponen-komponen fungsi yang sering digunakan antara lain:

Fungsi	Keterangan
Baca dan Tampilkan Kata	Membaca dan menampilkan kata satu per satu
Hitung Kata	Menghitung karakter satu per satu

Semua komponen mesin kata dapat digunakan untuk aplikasi yang lebih besar.

Berikut adalah contoh mesin kata yang menggunakan teks berupa string (*array* karakter) (dalam bahasa algoritmik):

```
{menggunakan komponen-komponen mesin karakter }

CCKata : string
panjang_kata : integer

procedure STARTKATA(teks : string)
{inisialisasi awal pemrosesan mesin karakter}

   START(teks)
```

```
{end procedure}

procedure RESETKATA(teks : string)
{mengosongkan CCKata}

   panjang_kata <- 0

{end procedure}

procedure INCKATA(teks : string)
{maju satu karakter}

    RESETKATA(teks)

   if not EOP(teks) then

      while (CC <> ' ') and (CC <> '.') do

         CCKata_panjang_kata+1 <- CC
         panjang_kata <- panjang_kata + 1

         INC(teks)

      {end while}

   {end if}

{end procedure}

function EOPKATA(teks : string) : boolean
{mengembalikan status apakah sudah bertemu akhir teks atau akhir array}

   -> EOP(teks)

{end function}
```

Berikut adalah algoritma untuk proses pembacaan dan menampilkan kata satu per satu menggunakan mesin kata di atas (dalam bahasa algoritmik):

```
{program utama}

   teks : string

   input(teks)

   STARTKATA(teks)

   while (not EOPKATA(teks)) do

      INCKATA(teks)
      if panjang_kata > 0 then

         output(CCKata)

      {end if}

   {end while}
```

```
{end program utama}
```

Berikut adalah algoritma untuk proses penghitungan kata menggunakan mesin karakter di atas (dalam bahasa algoritmik):

```
{program utama}

  teks : string
  jumlah_kata : integer

  input(teks)

  STARTKATA(teks)

  jumlah_kata <- 0

  while (not EOPKATA(teks)) do

    INCKATA(teks)

    if panjang_kata > 0 then

         jumlah_kata <- jumlah_kata + 1

    {end if}

  {end while}

  output(jumlah_kata)

{end program utama}
```

Soal Latihan:

1. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan mesin kata untuk menghitung panjang kata rata-rata dari teks.
2. Buatlah algoritma dan program dengan karakter bahasa Pascal atau C menggunakan mesin kata untuk menghitung banyaknya kata yang berakhiran 's' pada teks.
3. Buatlah algoritma dan program dengan bahasa Pascal atau C menggunakan mesin karakter untuk menghitung banyaknya kata "dan" pada sebuah teks.
4. Buat mesin kata dengan teks membaca dari *file* teks.
5. Kerjakan soal nomor 1 sampai 3 dengan menggunakan *file* teks.

Daftar Pustaka

[1] Griffith, Arthur. 2002. GCC : The Complete Reference. McGraw-Hill Book Co.
[2] Jamsa, Kris. 1996. 1001 Tips C/C++. Penerbit ANDI : Yogyakarta.
[3] Kadir, Abdul. 2001. Pemrograman C++. Penerbit ANDI : Yogyakarta.
[4] Liem, Inggriani. 2003. Diktat Kuliah IF1281 Algoritma dan Pemrograman. ITB : Bandung.
[5] Munir, Rinaldi. 2007. Algoritma dan Pemrograman. Penerbit Informatika: Bandung.
[6] Shalahuddin, M. dan Rosa A. S. 2007. Belanjar Pemrograman dengan Bahasa Pemrograman C++ dan Java: dari Nol Menjadi Andal. Penerbit Informatika: Bandung.

Lampiran - Prosedur dan Fungsi Standar pada Bahasa Pemrograman C

Bahasa Pemrograman C	
Matematika	
pustaka : math.h	
Prosedur/Fungsi	**Keterangan**
abs(bil_masukan) contoh : x = abs(y);	menjadikan bilangan bulat masukan sebagai bilangan absolut (positif)
fabs(bil_masukan) contoh : x = fabs(y);	menjadikan bilangan riil masukan sebagai bilangan absolut (positif)
sqrt(bil_masukan) contoh : x = sqrt(y);	untuk menghitung akar pangkat dua dari suatu bilangan positif
exp(bil_masukan) contoh : x = exp(y);	untuk menghitung $e^{bil\_masukan}$ dimana e adalah bilangan natural
log(bil_masukan) contoh : x = log(y);	untuk menghitung logaritma naturalis (berbasis e) dari bilangan masukan
log10(bil_masukan) contoh : x = log10(y);	untuk menghitung logaritma basis 10 dari bilangan masukan
pow(x,y)	untuk menghitung hasil x^y

contoh : z = pow(x,y);	
acos(bil_masukan) contoh : x = acos(y);	menghitung arcus cosinus bilangan masukan
asin(bil_masukan) contoh : x = asin(y);	menghitung arcus sinus bilangan masukan
atan(bil_masukan) contoh : x = atan(y);	menghitung arcus tangent bilangan masukan
cos(bil_masukan) contoh : x = cos(y);	untuk menghitung cosinus bilangan masukan
cosh(bil_masukan) contoh : x = cosh(y);	untuk menghitung cos hiperbolis dari bilangan masukan
sin(bil_masukan) contoh : x = sin(y);	untuk menghitung sinus dari bilangan masukan
sinh(bil_masukan) contoh : x = sinh(y);	untuk menghitung sinus hiperbolis dari bilangan masukan
tan(bil_masukan) contoh : x = tan(y);	untuk menghitung tangent bilangan masukan

tanh(bil_masukan) contoh : x = tanh(y);	untuk menghitung tangent hiperbolis dari bilangan masukan
atof(string_masukan) contoh : x = atof(“65.78”);	untuk mengubah string masukan menjadi bilangan riil, pustaka ditambah dengan stdlib.h
atoi(string_masukan) contoh : x = atoi(“23”);	untuk mengubah string masukan menjadi bilangan integer pustaka ditambah dengan stdlib.h
rand() contoh : x = rand();	menghasilkan bilangan bulat secara acak pustaka ditambah dengan stdlib.h
random(bil_masukan) contoh : x = random(100);	menghasilkan bilangan bulat secara acak diantara 0 sampai bilangan masukan dikurangi 1 pustaka ditambah dengan stdlib.h
randomize() contoh : randomize();	menginisialisasi proses pengacakan bilangan pustaka ditambah dengan stdlib.h dan time.h
srand(bil_masukan) contoh : srand(100);	menginisialisasi proses pengacakan mulai dari 0 sampai bilangan masukan dikurangi 1 pustaka ditambah dengan stdlib.h
max(a, b) contoh : float x = (float) max(a,b);	untuk menentukan bilangan terbesar antara a dan b
min(a,b)	untuk menentukan bilangan terkecil antara a dan b

contoh ; int x = (int) min(a,b);	
Manipulasi String	
pustaka :string.h	
Prosedur/Fungsi	**Keterangan**
strcpy(string_hasil, string_asal) contoh : strcpy(hasil, asal);	menyalin string asal ke string hasil
strcat(string_hasil, string_tambah) contoh : strcat(hasil, tambah);	menggabungkan isi string tambah ke string hasil

strcmp(string1, string2)
 	
contoh :
if(strcmp(string1, string2) == 0){
.....
}
	membandingkan dua buah string masukan dengan memperhatikan huruf besar dan huruf kecil
	
	menghasilkan bilangan negatif jika string1 lebih kecil dari string2, menghasilkan 0 jika kedua string sama, menghasilkan bilangan positif lebih besar 0 jika string1 lebih besar dari string2

strcmpi(string1, string2)
 	
contoh :
if(strcmpi(string1, string2) == 0){
.....
}
	membandingkan dua buah string masukan tanpa memperhatikan huruf besar dan huruf kecil
	
	menghasilkan bilangan negatif jika string1 lebih kecil dari string2, menghasilkan 0 jika kedua string sama, menghasilkan bilangan positif lebih besar 0 jika string1 lebih besar dari string2

strlen(string_masukan) contoh : x = strlen("program");	menghitung panjang karakter pada string masukan
strstr(string1, string2) contoh : strstr(string1, string2);	menentukan ada atau tidaknya suatu string2 di dalam string1 jika tidak ada maka hasilnya adalah null
tolower (string_masukan) contoh : tolower(teks);	mengubah semua karakter pada string masukan menjadi berhuruf kecil semua perlu ditambahkan pustaka ctype.h

strupr(string_masukan)
 	
contoh :
char teks[] = "informatika";
strupr(teks);
 	
atau
 	
toupper(string_masukan)
 	
contoh :
toupper(teks);
	mengubah semua karakter pada string masukan menjadi berhuruf besar semua
	
	jika menggunakan toupper() perlu ditambahkan pustaka ctype.h

Waktu	
pustaka : time.h	
Prosedur/Fungsi	**Keterangan**
_strdate(string_masukan) contoh : char tanggal[9]; _strdate(tanggal);	mengembalikan tanggal saat ini
_strtime(string_masukan) contoh : char waktu[9]; _strtime(waktu);	mengembalikan waktu saat ini