September
2014

PRODI TEKNIK INFORMATIKA

universitas
MALIKUSSALEH

BAHAN AJAR

SAFWANDI, ST

Struktur Data

FAKULTAS TEKNIK

Universitas Malikussaleh

Jurusan Teknik Kimia
Jurusan Teknik Industri
Jurusan Teknik Mesin
Jurusan Teknik Elektro
Jurusan Teknik Slipil
Prodi Teknik Informatika
ProdiTeknik Arsitektur

BAHAN AJAR

STRUKTUR DATA

BAHAN AJAR

Diterbitkan oleh
FAKULTAS TEKNIK UNIVERSITAS MALIKUSSALEH
PRODI TEKNIK INFORMATIKA

Alamat
Fakultas Teknik Universitas Malikussaleh
Jl. Cot Tengku Nie, Reuleut, Muara Batu,
Aceh Utara, Provinsi Aceh

BAHAN AJAR

(PRODI TEKNIK INFORMATIKA)

universitas
MALIKUSSALEH

STRUKTUR DATA

Disusun Oleh:

Safwandi, ST

Nurdin, S.Kom,. M.Kom

FAKULTAS TEKNIK
UNIVERSITAS MALIKUSSALEH
2014

universitas
MALIKUSSALEH

BAHAN AJAR

PRODI TEKNIK INFORMATIKA

TIM PENGELOLA BAHAN AJAR

FAKULTAS TEKNIK UNIVERSITAS MALIKUSSALEH

PENASEHAT:

Ir. T. Hafli,. MT
Dekan Fakultas Teknik Universitas Malikussaleh

PENANGGUNG JAWAB:

Herman Fithra, ST,. MT
Pembantu Dekan I Bidang Akademik

Bustami, S.Si,. M.Si
Pembantu Dekan II Bidang Keuangan

Edzwarsyah, ST,. MT
Pembantu Dekan III Bidang Kemahasiswaan

Salwin, ST,. MT
Pembantu Dekan IV Bidang Kerjasama dan Informasi

KETUA PENYUNTING:

Nurdin, S. Kom,. M. Kom
Ketua Prodi Teknik Informatika

Mukti Qamal, ST,. M.IT
Sekretaris Prodi Teknik Informatika

TATA USAHA DAN BENDAHARA:

Elizar, S. Sos
Kepala Tata Usaha

Ismail, ST
Bendahara

SAMBUTAN
KETUA PRODI TEKNIK INFORMATIKA
FAKULTAS TEKNIK UNIVERSITAS MALIKUSSALEH

Salah satu upaya untuk meningkatkan mutu pembelajaran sesuai Tri Dharma perguruan tinggi adalah penyusunan dan penyediaan bahan ajar. Struktur Data merupakan salah satu mata kuliah wajib dan juga salah satu cabang dari informatika yang begitu cepat berkembang. Untuk memfasilitasi standarisasi minimal perkuliahan Struktur Data di program studi teknik informatika, maka diperlukan bahan ajar yang tepat, aktual, dan memenuhi standar kurikulum nasional informatika.

Bahan ajar Struktur Data ini disusun skematis dengan materinya merupakan materi-materi standar pengajaran di hampir seluruh universitas yang menyelenggarakan perkuliahan Struktur Data. Dengan selesai disusunnya bahan ajar ini, maka kami berharap segala materi yang disuguhkan dapat memicu dan memotivasi para mahasiswa untuk lebih tangguh memperdalam teknik-teknik pemrograman yang sesuai standarisasi struktur data.

Reuleut, 25 Juli 2014
Ketua Program Studi Teknik Informatika

Nurdin, S.Kom,. M.Kom
Nip. 19781020 200501 1 003

KATA PENGANTAR

Dengan nama Allah SWT, yang telah memberi saya petunjuk dan pencerahannya sehingga tersusunnya bahan ajar Struktur Data ini.

Struktur data merupakan salah satu matakuliah dalam cabang ilmu komputer / Teknik Informatika yang paling pesat berkembang dan hampir selalu diaplikasikan untuk membantu bidang-bidang ilmu lainnya. Dengan mempelajari Struktur data mahasiswa dapat mengorganisir data menggunakan konsep struktur data dan mampu mengimplementasikannya ke dalam program.

Bahan ajar struktur data yang disusun ini terdiri dari delapan bab, yaitu : tipe data, rekursi, tumpukan, antrian, linked list, pointer, pengurutan, pencarian. Materi di dalam bahan ajar ini disampaikan secara sistematis serta disertai dengan contoh program pada setiap pakok bahasan dan disusun dengan penggunaan kalimat-kalimat yang sederhana dan pastinya representatif dari pengalaman penulis dalam mengampu mata kuliah struktur data.

Akhir kata penulis mengucapkan selamat membaca, dan atas segala kritik dan saran, penulis mengucapkan terima kasih.

Reuleut, 25 Juli 2014
Penulis

Safwandi, ST
Nip. 197712132008121004

LEMBARAN PENGESAHAN

This page is intentionallly left blank

DAFTAR ISI

SILABUS MATA KULIAH

Kode MK	:	TIF- 1332
Nama MK	:	Struktur Data
Bobot SKS	:	2
Tujuan	:	Memberikan pengetahuan tentang konsep pengorganisasian kumpulan data dan algoritma pemrograman struktur data dalam pemrograman
Prasyarat	:	Algoritma dan Pemrograman
Materi	:	Pengantar dan ruang lingkup Tipe data, Rekursi, Tumpukan, Antrian, Linked list, Pointer, Pengurutan, Pencarian dan Graphs.
Referensi	:	1. Ir. P. Insap santoso, M.Sc, Struktur data dengan Turbo Pascal Versi 6.0, Andi Offset Yogyakarta 2. D, Suryadi H.S,. Pengantar Struktur Data, Penerbit Gunadarma. 3. Loomis, Mary E.S,. Data Management and File Structures, Prentice Hall International Inc,. 1989. 4. Reynolds, W. Charles, Program Design and Data Structures in Pascal, Wadsworth Pub. Co,. 1986.

This page is intentionallly left blank

SATUAN ACARA PENGAJARAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 50 menit
Pertemuan ke : I

A. Tujuan Pembelajaran Umum (TPU) : Mahasiswa mampu memahami definisi dan bentuk umum Tipe data, seperti tipe data Array, Tipe data Record dalam pemrograman

B. Tujuan Pembelajaran Khusus (TPK)
1. Mahasiswa dapat menjelaskan definisi dan bentuk umum tipe data
2. Mahasiswa dapat menjelaskan bentuk deklarasi tipe data array dan contoh program array
3. Mahasiswa dapat menjelaskan bentuk deklarasi tipe data record dan contoh program record

C. Pokok Pembahasan : Tipe Data

D. Sub Pokok Pembahasan
1. Tipe Data
2. Array (Larik)
3. Record (Rekaman)

E. Kegiatan Belajar Mengajar

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan definisi dan bentuk Tipe Data 2. Menjelaskan bentuk dan deklarasi array serta contoh program 3. Menjelaskan bentuk dan deklarasi record serta contoh program	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal Tugas yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : II, III

A. Tujuan Pembelajaran Umum (TPU) : Mahasiswa mampu memahami pengertian Rekursi, Proses rekursi, Fungsi Fibonacci dan Permutasi.

B. Tujuan Pembelajaran Khusus (TPK)

1. Mahasiswa dapat menjelaskan definisi Rekursi dan contoh program
2. Mahasiswa dapat menjelaskan proses Rekursi dalam pemrograman
3. Mahasiswa dapat menjelaskan fungsi Fibonacci dalam pemrograman
4. Mahasiswa dapat menjelaskan Permutasi dan contoh program

C. Pokok Pembahasan : Rekursi

D. Sub Pokok Pembahasan

1. Pengertian Rekursi
2. Proses Rekursi
3. Fungsi Fibonacci
4. Permutasi

E. Kegiatan Belajar Mengajar

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	3. Menjelaskan definisi Rekursi dan contoh aplikasi 4. Menjelaskan Proses Rekursi 5. Menjelaskan Fungsi Fibonacci dan contoh program 6. Menjelaskan Permutasi dan contoh program	Memperhatikan/ bertanya	Projektor, white board
Penutup	7. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 8. Memberikan soal-soal kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : IV, V

A.	Tujuan Pembelajaran Umum (TPU)	:	Mahasiswa mampu memahami pengertian tumpukan, penyajian tumpukan, operasi pada tumpukan, dan contoh pemakaian tumpukan
B.	Tujuan Pembelajaran Khusus (TPK)A		1. Mahasiswa dapat menjelaskan pengertian Tumpukan. 2. Mahasiswa mampu menjelaskan proses dan penyajian tumpukan 3. Mahasiswa mampu menjelaskan operasi pada tumpukan 4. Mahasiswa mampu menjelaskan contoh pemakaian tumpukan dalam pemrograman
C.	Pokok Pembahasan		Tumpukan (Stack)
D.	Sub Pokok Pembahasan		1. Pengertian Tumpukan 2. Penyajian Tumpukan 3. Operasi pada tumpukan 4. Contoh pemakaian tumpukan
E.	Kegiatan Belajar Mengajar		

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian Tumpukan. 2. Menjelaskan proses dan penyajian tumpukan 3. Menjelaskan operasi pada tumpukan. 4. Menjelaskan contoh pemakaian tumpukan dalam program	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan soal-soal kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 50 menit
Pertemuan ke : VI

A. Tujuan Pembelajaran Umum (TPU) : Mahasiswa mampu memahami pengertian Antrian, Implementasi antrian dan operasi pada antrian

B. Tujuan Pembelajaran Khusus (TPK)
   1. Mahasiswa dapat menjelaskan pengertian Antrian.
   2. Mahasiswa mampu mengimplementasikan antrian dan menjelaskan operasi pada antrian

C. Pokok Pembahasan : Antrian

D. Sub Pokok Pembahasan
   1. Pengertian Antrian
   2. Implementasi Antrian

E. Kegiatan Belajar Mengajar

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian Antrian 2. Menjelaskan implementasi antrian dan operasi pada antrian	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan soal-soal kasus yang dikerjakan di	Memperhatikan/ Mengerjakan tugas	Projektor, white board

	rumah.		

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : VII, VIII

A. Tujuan Pembelajaran Umum (TPU) : Mahasiswa mampu memahami pengertian linked list, operasi pada linked list dan penyajian linked list

B. Tujuan Pembelajaran Khusus (TPK)
1. Mahasiswa dapat menjelaskan pengertian linked list
2. Mahasiswa mampu menjelaskan operasi pada linked list
3. Mahasiswa mampu menyajikan linked list dalam pemrograman

C. Pokok Pembahasan : Linked List

D. Sub Pokok Pembahasan
1. Pengertian linked list
2. Operasi pada linked list
3. Penyajian linked list

E. Kegiatan Belajar Mengajar

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian linked list 2. Menjelaskan operasi pada linked list	Memperhatikan/ bertanya	Projektor, white board

	3. Menjelaskan penyajian linked list dalam pemrograman		
Penutup	1. Memberikan kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : IX, X

A. Tujuan Pembelajaran Umum (TPU) : Mahasiswa mampu memahami pengertian pointer, deklarasi pointer dan alokasi tempat, serta operasi pada pointer dan contoh program

B. Tujuan Pembelajaran Khusus (TPK)
1. Mahasiswa dapat menjelaskan pengertian pointer
2. Mahasiswa mampu mendeklarasikan dan alokasi tempat
3. Mahasiswa dapat menjelaskan operasi pada pointer dalam pemrograman

C. Pokok Pembahasan : Pointer

D. Sub Pokok Pembahasan
1. Pengertian Pointer
2. Deklarasi pointer dan alokasi tempat
3. Operasi pada pointer

E. Kegiatan Belajar Mengajar

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian pointer 2. Menjelaskan deklarasi pointer dan alokasi tempat 3. Menjelaskan operasi pada pointer	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan soal-soal kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : XI, XII

A.	Tujuan Pembelajaran Umum (TPU)	:	Mahasiswa mampu menjelaskan pengertian pengurutan, pengurutan larik, metode pengurutan dan contoh program pengurutan.
B.	Tujuan Pembelajaran Khusus (TPK)		1. Mahasiswa dapat menjelaskan pengertian pengurutan 2. Mahasiswa dapat menjelaskan pengurutan larik 3. Mahasiswa dapat menjelaskan metode pengurutan dan contoh program pengurutan
C.	Pokok Pembahasan		Pengurutan
D.	Sub Pokok Pembahasan		1. Pengertian Pengurutan 2. Pengurutan larik 3. Metode Pengurutan 4. Contoh program pengurutan
E.	Kegiatan Belajar Mengajar		

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian pengurutan 2. Menjelaskan contoh pengurutan larik 3. Menjelaskan metode dalam pengurutan data 4. Menjelaskan contoh program pengurutan	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan soal-soal kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

SATUAN ACARA PERKULIAHAN (SAP)

Mata Kuliah : Struktur Data
Kode Mata Kuliah : TIF- 1332
Jumlah SKS : 2 (Dua)
Waktu Pertemuan : 2 x 2 x 50 menit
Pertemuan ke : XIII, XVI

A.	Tujuan Pembelajaran Umum (TPU)	:	Mahasiswa mampu menjelaskan pengertian pencarian, pencarian berurutan, pencarian biner dan contoh program pencarian.
B.	Tujuan Pembelajaran Khusus (TPK)		1. Mahasiswa dapat menjelaskan pengertian pencarian 2. Mahasiswa dapat menjelaskan pencarian berurutan 3. Mahasiswa dapat menjelaskan pencarian biner dan contoh program pencarian
C.	Pokok Pembahasan		Pencarian
D.	Sub Pokok Pembahasan		1. Pengertian pencarian 2. Pencarian berurutan 3. Pencarian Biner 4. Contoh program pencarian
E.	Kegiatan Belajar Mengajar		

Tahap	Kegiatan Dosen	Kegiatan Mahasiswa	Media dan Alat Pengajaran
Pendahuluan	1. Menjelaskan TPU dan TPK. 2. Memberikan referensi.	Memperhatikan/ bertanya	Projektor, white board
Penyajian	1. Menjelaskan pengertian pencarian 2. Menjelaskan pencarian berurutan 3. Menjelaskan pencarian biner 4. Menjelaskan contoh program pencarian	Memperhatikan/ bertanya	Projektor, white board
Penutup	1. Memberikan soal-soal kasus yang dikerjakan di kelas perkuliahan. 2. Memberikan soal-soal kasus yang dikerjakan di rumah.	Memperhatikan/ Mengerjakan tugas	Projektor, white board

BAB 1
TIPE DATA

PENDAHULUAN

Deskripsi Singkat

Materi perkuliahan meliputi definisi dan bentuk umum Tipe data, seperti tipe data Array, Tipe data Record dan contoh program array dan record

Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi dan bentuk umum Tipe data, seperti tipe data Array, Tipe data Record dan contoh program array dan record.

PENYAJIAN

1.1 Tipe Data

Dalam bahasa pemrograman pascal, semua perubah yang akan dipakai harus sudah ditentukan tipe datanya. Dengan menentukan tipe data suatu perubah, sekaligus menentukan batasan nilai perubah tersebut dan jenis operasi yang bisa dilaksanakan atas perubah tersebut.

Bentuk umum dari deklarasi tipe data adalah:

Type pengenal = tipe;

Dengan, Pengenal : nama pengenal yang menyatakan tipe data.
Tipe : Tipe data yang digunakan.

1.2 Array (*Larik*)

Array adalah tipe terstruktur yang mempunyai komponen dalam jumlah yang tetap dan setiap komponen mempunyai tipe data yang sama. Posisi masing-masing komponen dalam array dinyatakan sebagai nomor index. Dalam bentuknya array dapat kita tinjau dari segi pengaturan struktur datanya dalam konteks dimensi sebagai berikut:

1. Array 1-Dimensi, contoh: List, Vektor
2. Array 2-Dimensi, contoh: Tabel, Matriks (2 dimensi)
3. Array 3-Dimensi, contoh: Matriks 3 dimensi

Bentuk umum dari deklarasi tipe array adalah:

Var nama : **array** [index] **of** tipe

Keterangan:

Var, array, of	:	Kata baku yang harus ada
Nama	:	nama array yang akan dideklarasikan
Index	:	cacah elemen pada array (batas nilai awal dan nilai akhir array)
Type	:	Tipe array, (Tipe array bisa sembarang tipe)

Contoh pendeklarasian array:

Var gaji : array [1..10] of real;

Artinya mendeklarasikan gaji yang bertipe real, mempunyai 10 elemen dengan indexnya menggunakan subjangkauan integer, yang mempunyai batas bawah 1 dan batas atas 10.

Contoh pendeklarasian array dimensi Satu:

Seorang mahasiswa mempunyai 3 buah nilai, maka deklarasinya:

```
Var data_nilai: array [1..3] of byte;
```

Contoh pendeklarasian array multi dimensi:

Ada 4 mahasiswa, dan tiap mahasiswa mempunyai 3 buah nilai, maka deklarasinya:

```
Var data_nilai: array [1..4] of array [1..3] of byte;
```

Atau disederhanakan menjadi,

```
Var data_nilai: array [1..4,1..3] of byte;
```

Contoh kasus:

Seorang mahasiswa mengambil 8 matakuliah, nilai yang diperoleh berupa angka (0 – 100). Buatlah program untuk menghitung nilai rata-ratanya

Penyelesaian:

```
Program rata_rata;
Uses crt;
    Var nilai_test: array [1..8] of byte;
    I, Jumlah: integer;
    Rata: real;
Begin
Clrscr;
    Jumlah:=0;
    For I:= 1 to 8 do
    Begin
      Write ('Nilai ke', I); Readln(nilai_test[I]);
      Jumlah:=Jumlah + nilai_test[I]
    End;
      Rata:=Jumlah/8;
      Write ('Nilai Rata-rata',Rata);
```

```
    Readln;
End.
```

1.3 Rekaman (*Record*)

Sama halnya dengan array, rekaman (record) adalah kumpulan data. Perbedaan antara array dengan record adalah bahwa dalam array semua elemennya harus bertipe sama, tetapi dalam rekaman setiap elemen bisa mempunyai tipe data yang berbeda satu sama lain. Dalam aktifitas sehari-hari pemakaian rekaman lebih banyak digunakan dibandingkan dengan array. Beberapa contoh pemakaian, misalnya rekaman data akademis mahasiswa, rekaman gaji pegawai, persediaan barang dalam gudang, dll.

Bentuk umum deklarasi rekaman (record) adalah:

```
Tipe pengenal = record
                  Medan1 : tipe1;
                  Medan2 : tipe2;
                  ….. …….
                  Medan n : tipe n;
                  end;
```

Dengan:

pengenal : pengenal yang menunjukan tipe data yang akan dideklarasikan.

Medan1,…,..Medan n : nama medan yang akan digunakan.

Tipe1,……,tipe n : sembarang tipe data yang telah dideklarasikan sebelumnya.

Berikut adalah contoh deklarasi rekaman.

```
Type tgl_kalender = record
                      Tanggal :1..31;
                      Bulan  : 1..12;
                      Tahun  : 1900..2010
                    end;

    Siswa = record
              Nama    : String [25];
              Alamat  : String [35];
              Kelamin : (L,P);
              Kelas   : 1..6;
            end;
```

Untuk memanipulasi medan pada suatu rekaman, harus ditulis dengan menggunakan bentuk umum:

Nama_rekaman . nama_medan

Notasi di atas disebut penanda medan (field designator). Sebagai contoh, untuk rekaman bertipe siswa yang dideklarasikan pada contoh diatas dan deklarasi:

```
Var murid : Siswa;
```

Kita bisa membaca medan nama dan alamat menggunakan statement :

```
Readln (murid . Nama);
Readln (murid . Alamat);
```

Ada cara yang lebih singkat dari cara diatas, khususnya jika harus memasuk sejumlah medan dalam saat yang bersamaan. Untuk itu kita bisa menggunakan statement **with**.
Bentuk umum statement with adalah:

with nama_rekaman do

Dengan nama_rekaman adalah nama rekaman yang akan dimasuk, dengan menggunakan deklarasi rekaman bertipe siswa, dan statement with, maka kita bisa memasup rekaman murid sebagai berikut:

```
with murid do
  Begin
     Readln (Nama);
     Readln (Alamat);
     Readln (Kelas);
     Readln (Kode_sex);
     If Kode_sex = 1 then Kelamin := L
     Else Kelamin := P
  end;
```

Cara diatas akan sama hasilnya jika anda tulis secara lengkap sebagai:

```
Readln (murid.Nama);
Readln (murid.Alamat);
Readln (murid.Kelas);
Readln (murid.Kode_sex);
If Kode_sex = 1 then murid.Kelamin := L
Else murid.Kelamin := P
```

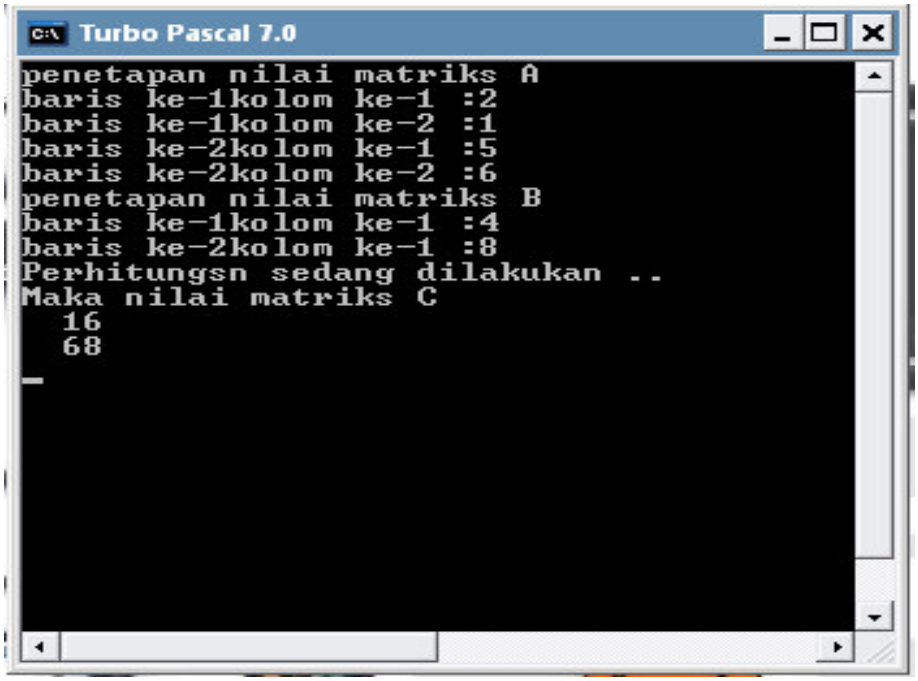
1.4 Contoh program Array dan Record

Contoh Program menggunakan tipe data Array

```
program array_perkalian_matriks;
uses crt;
const n_i=2;
n_j=2;
n_k=1;
var
A: array [1..n_i,1..n_j] of integer;
B: array [1..n_j,1..n_k] of integer;
```

```
C: array [1..n_i,1..n_k] of integer;
i,j,k:integer;
begin
clrscr;
writeln('penetapan nilai matriks A');
for i:=1 to n_i do
for j:=1 to n_j do
begin
write('baris ke-',i,'kolom ke-',j,' :');
readln(A[i,j]);
end;
writeln('penetapan nilai matriks B');
for j:=1 to n_j do
for k:=1 to n_k do
begin
write('baris ke-',j,'kolom ke-',k,' :');
readln(B[j,k]);
end;
writeln('Perhitungsn sedang dilakukan ..');
for i:= 1 to n_i do
for k:= 1 to n_k do
begin
C[i,k] :=0;
for j:= 1 to n_j do
C[i,k] := C[i,k]+A[i,j] * B[j,k];
end;
writeln('Maka nilai matriks C');
for i:=1 to n_i do
begin
for k:=1 to n_k do
write(C[i,k]:4);
writeln;
end;
readln;
end.
```

Hasil output program perkalian matrik



Gambar 1.1 output program perkalian matrik

Contoh Program menggunakan tipe data Record

```
Contoh Program_record ;
Uses
        WinCrt ;
Type
        Petunjuk_Pegawai = ^Data_Pegawai ;
        Data_Pegawai = Record
        Nip : String[9] ;
        Nama : String[25] ;
        Gaji : Real ;
End ;
        Var
        Data_Pegawaix : Array[1..20] of Petunjuk_Pegawai;
        i, j, n : Byte ;
        t1, t2 : String ;
        t3 : Real ;
Begin
        Write('Banyaknya data...? ') ; ReadLn(n);
        For i := 1 to n do
                Begin
                        Writeln;
                        Writeln('Data Pegawai ke ',i:2);
                        New(Data_Pegawaix[i]) ;
                        With Data_Pegawaix[i]^ do
                                Begin
                                        Write('N I P         : '); ReadLn(Nip) ;
```

```
                                Write('Nama Pegawai   : '); ReadLn(Nama) ;
                                Write('Gaji Pokok     : Rp '); ReadLn(Gaji) ;
                        End ;
                End ;
                ClrScr;
                WriteLn(' DATA PEGAWAI UNIMAL') ;
                WriteLn('-------------------------------------------------------------------------')
        ;
                WriteLn(' No.| N I P     | NAMA PEGAWAI            | GAJI POKOK ')
        ;
                WriteLn('-------------------------------------------------------------------------')
        ;
                For i := 1 to n do
                With Data_Pegawaix[i]^ do
                Writeln(i:3,' | ',Nip:9,' | ',Nama:25,' | Rp ',Gaji:8:2) ;
                For i := 1 to n do
                Begin
                        For j := i to n do
                                Begin
                                If Data_Pegawaix[i]^.Nip > Data_Pegawaix[j]^.Nip
                                then
                                Begin
                                t1 := Data_Pegawaix[i]^.Nip ;
                                Data_Pegawaix[i]^.Nip := Data_Pegawaix[j]^.Nip ;
                                Data_Pegawaix[j]^.Nip := t1 ;
                                t2 := Data_Pegawaix[i]^.Nama ;
                                Data_Pegawaix[i]^.Nama := Data_Pegawaix[j]^.Nama ;
                                Data_Pegawaix[j]^.Nama := t2 ;
                                t3 := Data_Pegawaix[i]^.Gaji ;
                                Data_Pegawaix[i]^.Gaji := Data_Pegawaix[j]^.Gaji ;
                                Data_Pegawaix[j]^.Gaji := t3 ;
                                End ;
                                End ;
                End ;
End. { Akhir program }
```

Hasil output program data pegawai

Gambar 1.2 output program data pegawai

PENUTUP

Tugas:

1. Ada 5 orang mahasiswa dan tiap mahasiswa mengambil 7 matakuliah, nilai yang diperoleh berupa angka (0 – 100). Buatlah program untuk mencari nilai rata-ratanya dengan menggunakan Variabel Array.
2. Buatlah program pendataan data mahasiswa dengan menggunakan tipe data record.

BAB 2
REKURSI

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi rekursi, proses rekursi, fungsi fibonacci, menyusun permutasi dan disertai dengan contoh program rekursi.

Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi rekursi, proses rekursi, fungsi fibonacci, menyusun permutasi dan dapat membuat program rekursi.

PENYAJIAN

2.1 Pengertian Rekursi

Salah satu keistimewaan yang dimiliki Pascal adalah bahwa Pascal dapat melakukan suatu proses yang disebut dengan proses rekursi. Sifat rekursif ini dimiliki oleh beberapa statmen Pascal. Rekursi berarti suatu proses dapat memanggil dirinya sendiri. Dalam rekursi sebenarnya terkandung pengertian prosedur atau fungsi. Perbedaan adalah bahwa rekursi dapat memanggil dirinya sendiri, tetapi prosedur atau fungsi harus dipanggil lewat pemanggil prosedur atau fungsi. Rekursi merupakan teknik pemrograman yang penting, dan beberapa bahasa pemrograman modern mendukung keberadaan proses rekursi ini. Dalam prosedur atau fungsi, pemanggilan ke dirinya sendiri bisa berarti proses berulang yang tidak dapat diketahui kapan akan berakhir. Dalam pemakaian sehari-hari, rekursi merupakan teknik pemrograman yang berdayaguna untuk digunakan pada pekerjaan pemrograman dengan mengeksresikannya ke dalam suku-suku dari program lain dengan menambahkan langkah-langkah sejenis.

Contoh paling sederhana dari proses rekursi adalah proses menghitung nilai faktorial dari bilangan bulat positif dan mencari deret Fibonacci dari suatu bilangan bulat.
Nilai faktorial, secara rekursif dapat ditulis sebagai:

0! = 1

N! = N x (N-1)! Untuk N > 0

Yang secara notasi pemrograman dapat ditulis sebagai:

Faktorial (0) = 1 ..... (1)

Faktorial (N) = N * faktorial (N-1) ..... (2)

Persamaan (2) di atas merupakan contoh hubungan rekurens (recurrence relation), yang berarti bahwa nilai suatu fungsi dengan argumen tetentu dapat dihitung dari fungsi yang sama dengan argumen yang lebih kecil. Persamaan (1) yang tidak bersifat rekursif, disebut nilai awal. Setiap fungsi rekursi paling sedikit mempunyai 1 nilai awal: jika tidak, fungsi tersebut tidak dapat dihitung secara eksplisit. Bilangan Fibonacci dapat didefinisikan berdasarkan deret integer tak terhingga sebagai berikut:

1 1 2 3 5 8 13 21 34 55 89

dari deret di atas dapat dilihat bahwa bilangan ke N (N > 2) dalam deret dapat dicari dari 2 bilangan sebelumnya yang terdekat dengan bilangan ke N, yaitu bilangan ke (N-1) dan bilangan ke (N-2). Sehingga, jika FIBO (N) menunjukan bilangan Fibonacci ke N, maka FIBO (N) dapat dihitung berdasar hubungan rekurens:

FIBO (N) = FIBO (N-1) + FIBO (N-2)

Karena FIBO (n) ditentukan oleh dua nilai yang berbeda dengan argumen yang nilainya lebih kecil, maka untuk mencari bilangan Fibonacci diperlukan dua nilai awal, yaitu:

FIBO (1) = 1 dan FIBO (2) = 2

2.2 Proses Rekursif

Untuk memahami proses rekursi yang terjadi dalam sebuah fungsi rekursi, perhatikan contoh sederhana di bawah ini, yang menyajikan satu fungsi untuk menghitung harga faktorial suatu integer berdasarkan hubungan rekurens seperti yang dijelaskan di atas.

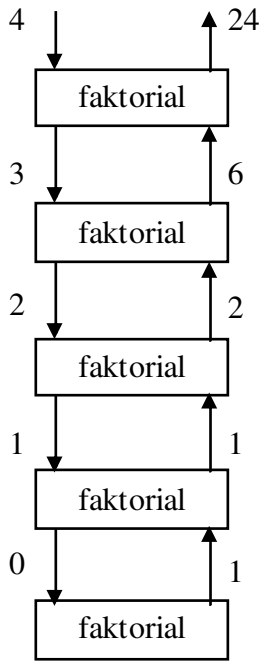
```
Function faktorial (N : integer): integer;
Begin
    If n = 0 then
        Faktorial := 1;
    Else
        Faktorial := n * faktorial (n-1)
```

```
End;
program menghitung_faktorial;
uses
   crt;
var
  i,n:integer;
function faktorial (n:integer):integer;
begin
    if n = 0 then
       faktorial := 1
    else
        faktorial := n * faktorial (n-1)
end;
begin
    clrscr;
    write('masukan nilai integer = ');
    read (i);
    n := faktorial (i);
    writeln ('Faktorial ',i,' adalah ',n);
    readln;readln;
end.
```

Output program tersebut adalah sebagai berikut:

```
masukan nilai integer = 4
Faktorial  4  adalah  24
```

Dari fungsi di atas dapat dilihat bahwa faktorial (N) dapat dihitung dari faktorial (N-1), dimana faktorial (N-1) dapat dihitung dari faktorial (N-2) dan seterusnya. Dengan demikian, fungsi di atas untuk N=4 dapat kita lacak cara kerjanya seperti terlihat pada Gambar berikut:



Gambar 2.1 faktorial.

Perhatikan bahwa untuk menghitung faktorial (N), maka fungsi harus memanggil nilai faktorial (N-1) yang telah diperoleh. Demikian juga untuk menghitung nilai faktorial (N-1), fungsi harus memanggil nilai faktorial (N-2), dan setrusnya. Notasi faktorial (N-1), yang digunakan untuk memanggil nilai fungsi sebelumnya, sering disebut dengan pemanggil rekursi atau (*recursion call*). Dalam ilustarasi di atas , pemanggil rekursi ditunjukkan dengan tanda anak panah.

Sama halnya dengan prosedur atau fungsi yang lain, maka dalam rekursi pasti ada pemanggil rekursi. Gambar tersebut di atas dapat dijelaskan sebagai berikut: (panah ke bawah menunjukkan nilai parameter aktual yang dilewatkan ke dalam fungsi faktorial). Pada saat Turbo Pascal mengerjakan faktorial (4) ia akan mentest bahwa 4 tidak sama dengan 0 (karena faktorial (0) diterapkan sebagai nilai awal).

Dengan demikian, ia perlu menghitung faktorial (N-1), yang sama dengan faktorial (3), terlebih dahulu, lihat panah ke bawah dengan garis ganda. Sehingga Turbo Pascal akan mengerjakan fatorial (3), sementara bilangan 4 dan operatornya akan disimpan lebih dahulu. Dalam hal inipun Turbo Pascal menjumpai bahwa 3 tidak sama dengan 0. Proses diteruskan sampai Turbo Pascal mengeksekusi faktorial (0), yaitu nilai awal yang ditentukan, maka proses rekursif dihentikan. Dengan telah diperolehnya nilai faktorial (0), maka segera dapat diperoleh nilai faktorial (1) yang sama dengan 1 * faktorial (0), yang memberi hasil sama dengan 1 (sekarang Anda dipersilahkan untuk melihat panah ke atas). Proses ini diteruskan sampai dihasilkan nilai terakhir yang dikembalikan oleh fungsi faktorial ini, yaitu sama dengan 24. Dengan melihat pada ilustrasi pada Gambar tersebut di atas, dapat diperhatikan bahwa angka di sebelah kiri menunjukan nilai parameter aktual yang dilewatkan ke fungsi faktorial, dan angka di sebelah kanan menunjukan nilai faktorial yang diperoleh setelah fungsi faktroial selesai dikerjakan menggunakan nilai parameter aktual di atas.

Jika diperhatikan kembali proses di atas, untuk menyimpan nilai N yang belum dioperasikan Turbo Pascal memanfaatkan struktur tumpukan (*stack*).

Dalam hal ini, jika Turbo Pascal belum mengeksekusi faktorial (0), maka nilai N saat itu akan dimasukkan ke dalam tumpukan (lihat anak panah ke bawah), dan akan dikeluarkan kembali jika prosesnya mengikuti arah anak panah ke atas seperti ditunjukkan dalam gambar tersebut di atas.

2.3 Fungsi Fibonacci

Dengan cara yang sama, Anda dapat melacak proses yang terjadi pada saat Turbo Pascal mengeksekusi fungsi FIBO untuk nilai N tertentu.

Fungsi FIBO dapat diimplementasikan seperti yang tersaji dalam program berikut:

```
Function FIBO (n:integer):integer;
Begin
        If (n = 1) or (n = 2) then
        FIBO := 1;
        Else
                FIBO := FIBO (n-1) + FIBO (n-2)
End;
```

Program yang digunakan untuk menghitung fungsi Fibonacci adalah sebagai berikut:

```
program Fungsi_Fibonacci;
uses
   crt;
var
  i,n : integer;
function FIBO (n : integer): integer;
begin
    if (n = 1 ) or (n = 2) then
    FIBO := 1
    else
       FIBO := FIBO (n-1) + FIBO (n-2)
end;
begin
    clrscr;
```

```
    write('Masukan Nilai Integer : '); read(i);
    n:=FIBO(i);
    write('Nilai Fibonacci : ',i:2,' adalah : ', n:4);
    readln;readln;
end.
```

Hasil yang didapatkan dari eksekusi program tersebut sebagai berikut:

```
Masukan Nilai Integer : 3
Nilai Fibonacci :  3 adalah :    2
1  1  2
-----------------------------------------
Masukan Nilai Integer : 4
Nilai Fibonacci :  4 adalah :    3
1  1  2  3
-----------------------------------------
Masukan Nilai Integer : 5
Nilai Fibonacci :  5 adalah :    5
1  1  2  3  5
-----------------------------------------
Masukan Nilai Integer : 6
Nilai Fibonacci :  6 adalah :    8
1  1  2  3  5  8
-----------------------------------------
Masukan Nilai Integer : 7
Nilai Fibonacci :  7 adalah :   13
1  1  2  3  5  8  13
-----------------------------------------
Masukan Nilai Integer : 8
Nilai Fibonacci :  8 adalah :   21
1  1  2  3  5  8  13  21
-----------------------------------------
Masukan Nilai Integer : 9
Nilai Fibonacci :  9 adalah :   34
1  1  2  3  5  8  13  21  34
```

```
----------------------------------------
Masukan Nilai Integer : 10
Nilai Fibonacci : 10 adalah :  55
1  1  2  3  5  8  13  21  34  55
----------------------------------------
```

Rekursi Versus Iterasi

Dalam beberapa hal, rekursi kurang efisien dibandingkan dengan proses iterasi. Bandingkan fungsi FIBO di atas dengan fungsi di bawah ini yang juga digunakan untuk menghitung bilangan Fibonacci, tetapi dilaksanakan dengan cara iterasi seperti berikut:

```
function FIBO_ITERASI (n: integer) : integer;
var
    akhir, bantu, f, i : integer;
begin
    i:=1; f:=1; akhir:=0;
    if n= 0 then f:=0;
    while i <> n do
    begin
        bantu := f; i := i+1;
        f := f+akhir;
        akhir := bantu
    end;
    FIBO_ITERASI := f
End;
```

Dalam contoh pertama (fungsi FIBO yang menggunakan proses rekursif), suatu bilangan pada suku ke n akan diperoleh melalui hubungan rekurens. Sebaliknya, cara rekursif di atas adalah kurang nyata, karena ada 2 pemanggil rekursif diikuti yang lain. Meskipun demikian, cara rekursif di atas menjadi terlalu buruk, karena kedalaman pemanggil rekursif sangat diperlukan dan juga karena pengulangan penghitungan dari suatu hasil yang sebenarnya telah dihitung. Sebagai contoh, FIBO (6) memerlukan pemanggilan FIBO (5) dan FIBO (4), dimana FIBO (5) memerlukan pemanggilan FIBO (4) dan FIBO (3), dan seterusnya. Dalam beberapa situasi, pemecahan secara rekursif dan secara iterasi mempunyai keuntungan dan kekurangan yang dapat saling diperbandingkan. Adalah cukup sulit untuk menentukan mana yang paling sederhana, paling jelas, paling efisien dan paling mudah dibandingkan dengan yang lain. Dapat ditambahkan, pemilihan cara iterasi maupun cara rekursif boleh

dikatakan merupakan kesenagan seseorang progremmer sesuai dengan kesenangan dan cita rasanya sendiri.

Program selengkapnya dengan menggunakan iterasi Fibonacci adalah sebagi berikut:

```
program iterasi;
uses
   crt;
var
  f, akhir, bantu, i, n:integer;
function fibo_iterasi (n:integer):integer;
var
  f, akhir, bantu, i : integer;
begin
    i := 1; f := 1; akhir := 0;
    if n = 0 then f := 0;
    while i <> n do
    begin
        bantu := f; i := i+1;
        f := f + akhir;
        akhir := bantu
    end;
    fibo_iterasi := f
end;
begin
    clrscr;
    write('masukan nilai integer = ');
    read (i);
    n := fibo_iterasi (i);
    writeln ('Fibonacci dengan menggunakan iterasi untuk angka bulat ',i,'  adalah ',n);
    readln;readln;
end.
```

2.4 Menyusun permutasi

Contoh lain dari proses rekursif yang akan disajikan adalah untuk menyusun semua permutasi yang mungkin dari sekelompok karakter. Sebagai contoh, jika kita mempunyai 3 buah karakter A, B, dan C, maka semua permutasi yang mungkin dari ketiga karakter ini adalah:

A B C B A C C A B

A C B B C A C B A

Secara umum, banyaknya permutasi dari N buah karakter adalah N faktorial. Dalam contoh di atas N = 3, sehingga banyaknya permutasi adalah 3! = 6;
Proses penyusunan permutasi dapat dijelaskan sebagai berikut:

- cetak elemen ke-1, dan cetak permutasi elemen ke-2 sampai ke-N (permutasi dengan N-1 elemen)
- cetak elemen ke-2, dan cetak permutasi elemen ke-1, elemen ke-3 sampai ke-N (permutasi dengan N-1 elemen)
- cetak elemen ke-3, dan cetak permutasi elemen ke-1, elemen ke-2, elemen ke-4 sampai ke-N (permutasi dengan N-1 elemen)
- dan seterusnya, sampai langkah terakhir adalah cetak elemen ke-N, dan cetak permutasi elemen ke-1 sampai elemen ke (N-1) (permutasi dengan N-1 elemen).

Proses di atas diulang terus sampai dicetak permutasi dengan 1 elemen. Sebagai contoh, untuk N=3, maka caranya adalah:
- cetak ‘A’, dan cetak permutasi (‘B’’C’);
- cetak ‘B’, dan cetak permutasi (‘A’’C’);
- cetak ‘C’, dan cetak permutasi (‘A’‘B’);

Dari contoh di atas dapat dilihat bahwa banyaknya elemen yang dipermutasikan makin lama makin sedikit sampai sama dengan 1.
Kondisi inilah yang kita pakai sebagai batas proses. Program selengkapnya tersaji di bawah ini.

```
program susun_permutasi;
uses
   crt;
const
    max = 5;
type
   larik = array [1..max] of char;
var
  a : larik;
  c_permutasi, c_elemen, i : integer;
  lagi : char;
{ prosedur penyesunan permutasi }
```

```
procedure permutasi (var b:integer; a:larik; k, n :integer);
var
  i    : integer;
  temp : char;
begin
    if k = n then
      begin
        b:=succ(b);
        writeln('Permutasi ke ', b:2, ' : ');
        for i:=1 to n do
           write(a[i]:3);
        writeln;
      end
    else
    for i:=k to n do
       begin
          temp:=a[i];
          a[i]:=a[k];
          a[k]:=temp;
          permutasi(b, a, k+1, n)
       end
end;
{ program utama }
begin
    repeat
        clrscr;
        write('banyaknya karakter yang akan dipermutasi : ');
repeat
            gotoxy(47, 1); write('    ');
            gotoxy(47, 1); readln (c_elemen)
        until
           c_elemen <= max;
        { menyusun karakter yang akan dipermutasikan }
        for i := 1 to c_elemen do
```

```
        a[i] := chr(i+64);
        clrscr;
        writeln('Menyusun permutasi untuk ',c_elemen, ' karakter');
        writeln('-----------------------------------:');
        writeln;
        { proses mencari permutasi }
        c_permutasi := 0;
        permutasi(c_permutasi, a, 1, c_elemen);
        { mencetak hasil }
        writeln;
        writeln('Banyaknya permutasi: ',c_permutasi:3);
        writeln;
        write('akan coba lagi ? (Y/T): '); readln(lagi)
    until not (lagi in ['Y','y'])
end.
```

Output program tersebut dapat disajikan sebagai berikut:

```
banyaknya karakter yang akan dipermutasi :    3
Menyusun permutasi untuk 3 karakter
-----------------------------------------------------------------:
Permutasi ke  1 :
 A  B  C
Permutasi ke  2 :
 A  C  B
Permutasi ke  3 :
 B  A  C
Permutasi ke  4 :
 B  C  A
Permutasi ke  5 :
 C  A  B
Permutasi ke  6 :
 C  B  A
Banyaknya permutasi:   6
akan coba lagi ? (Y/T): T
```

This page is intentionallly left blank

BAB 3
TUMPUKAN (STACK)

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi tumpukan, penyajian tumpukan, operasi pada tumpukan, contoh pemakaian tumpukan dan disertai dengan contoh program tumpukan.

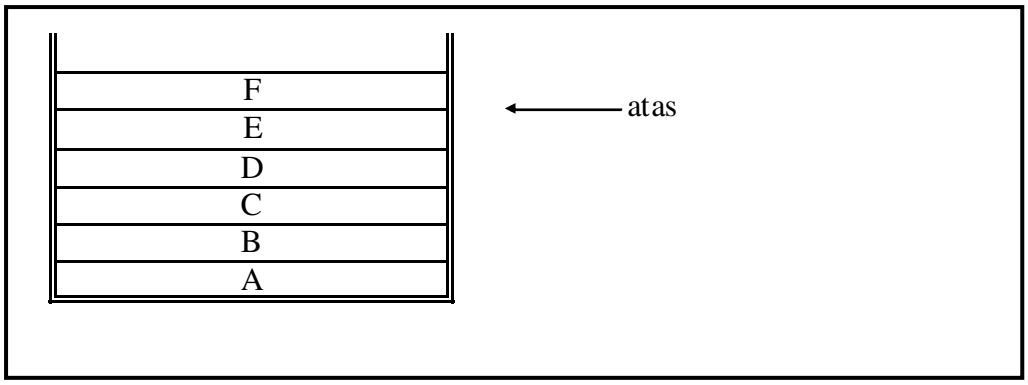
Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi tumpukan, operasi pada tumpukan, contoh pemakaian tumpukan dan dapat membuat program tumpukan.

PENYAJIAN

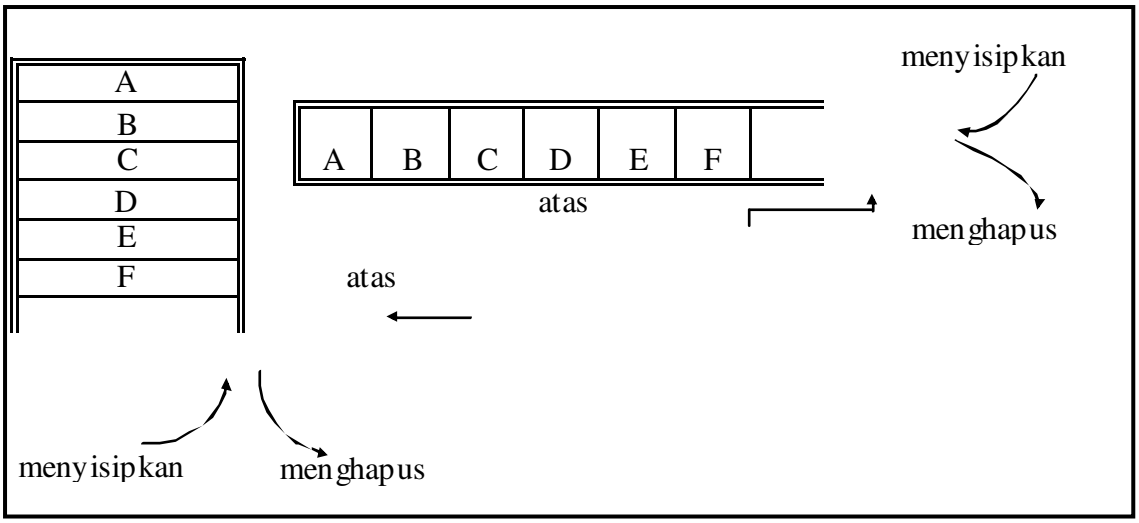
3.1 Pengertian Tumpukan

Secara sederhana, tumpukan dapat diartikan sebagai suatu kumpulan data yang seolah-olah ada data yang diletakkan di atas data yang lain. Satu hal yang perlu diingat adalah bahwa dapat ditambah (menyisipkan) data, dan mengambil (menghapus) data lewat ujung yang sama, yang disebut sebagai ujung atas tumpukan (*top of stack*). Untuk dijelaskan pengertian di atas diambil contoh sebagai berikut. Misalnya kita mempunyai dua buah kotak yang ditumpuk, sehingga kotak diletakkan di atas kotak yang lain. Jika kemudian tumpukan dua buah kotak itu ditambah dengan kotak ketiga, keempat dan seterusnya, maka akan diperoleh sebuah tumpukan kotak, yang terdiri dari N kotak. Secara sederhana, sebuah tumpukan dapat diilustrasikan pada Gambar 3.1 di bawah ini. Dari gambar ini dapat dikatakan bahwa kotak B ada di atas kotak A dan ada di bawah kotak C. Gambar di bawah ini menunjukkan bahwa dalam tumpukan hanya dapat ditambah atau diambil sebuah kotak lewat satu ujung, yaitu ujung bagian atas. Dapat dilihat pula bahwa tumpukan merupakan kumpulan data yang sifatnya dinamis, artinya dapat ditambah dan diambil data darinya.



Gambar 3.1 Tumpukan yang terdiri dari 6 kotak

Timbul pertanyaan, ujung yang manakah yang dianggap sebagai ujung atas tumpukan tersebut. Untuk menjawab pertanyaan ini harus ditentukan ujung mana yang kita gunakan untuk mengambil atau menyisipkan data yang baru. Dengan penggambaran tumpukan seperti Gambar 1 di atas, kita menganggap atau memilih bahwa kotak F adalah bagian atas dari tumpukan tersebut. Jika ada kotak lain yang akan disisipkan, maka ia akan diletakkan di atas kotak F, dan jika ada kotak yang akan diambil, maka kotak F lah yang akan diambil pertama kali. Penggambaran tumpukan tidak harus seperti di atas. Dapat digambar tumpukan dengan cara lain, seperti terlihat pada Gambar 3.2. Untuk mempermudah pemahaman, kita akan menggambarkan tumpukan seperti Gambar 3.1. Dengan memperhatikan ilustrasi yang disebutkan maka dapat dilihat bahwa tumpukan merupakan suatu senarai (*list*) yang mempunyai watak **"masuk terakhir keluar pertama"** (*last in first out* – LIFO).



Gambar 3.2 Cara lain penggambaran tumpukan

3.1.1 Penyajian Tumpukan

Sebelum kita melihat pada operasi dasar pada sebuah tumpukan, akan dilihat terlebih dahulu bagaimana menyajikan sebuah tumpukan dalam Bahasa Pemrograman Pascal. Ada beberapa cara untuk menyajikan sebuah tumpukan. Dalam subbagian ini

kita akan menggunakan cara yang paling sederhana. Seperti dijelaskan di atas, tumpukan adalah kumpulan data. Dalam Pascal sudah dikenal tipe data terstruktur yang disebut larik (*array*). Dengan demikian dapat digunakan larik ini untuk menyajikan sebuah tumpukan. Tetapi dapat segera dilihat bahwa penyajian tumpukan menggunakan larik adalah kurang tepat. Alasannya adalah bahwa banyaknya elemen dalam larik sudah tertentu (statis), sedangkan dalam tumpukan banyaknya elemen dapat sangat bervariasi (dinamis). Meskipun demikian, larik dapat digunakan untuk menyajikan sebuah tumpukan, dengan anggapan bahwa banyaknya elemen maksimum dari tumpukan tersebut tidak akan melebihi batas maksimum banyaknya elemen dalam larik. Pada suatu saat ukuran tumpukan akan sama dengan ukuran larik. Kalau kita teruskan menambah data lagi, akan terjadi *overflow*. Dengan demikian perlu data tambahan untuk mencatat posisi ujung tumpukan. Dengan kebutuhan seperti ini, dapat disajikan tumpukan menggunakan tipe data terstruktur yang lain, yaitu tipe rekaman (***record***) yang terdiri dari dua medan. Medan pertama bertipe larik untuk menyimpan elemen tumpukan, medan kedua bertipe **integer** untuk mencatat posisi ujung tumpukan. Dengan anggapan ini maka dapat dideklarasikan tumpukan seperti berikut ini:

```
const
        MaxElemen = 255;
type
        Tumpukan = record
                                Isi : array [1..MaxElemen] of integer;
                                Atas : 0..MaxElemen
                        end;
var
        T : Tumpukan;
```

Dengan deklarasi di atas dianggap bahwa elemen tumpukan T, yang tersimpan dalam larik T.Isi, adalah bertipe ***integer*** dan banyaknya elemen tumpukan maksimum adalah sebesar MaxElemen, yang dalam hal ini 255 elemen. Sesungguhnya elemen tumpukan tidak harus berupa ***integer***, tetapi dapat berupa data dengan tipe yang lain, misalnya ***real*** atau ***char***. Tetapi, tipe data dari medan **Atas** harus bilangan bulat antara 0 sampai MaxElemen, karena nilainya menunjukkan banyaknya elemen yang ada dalam suatu tumpukan, yang sekaligus menunjukkan posisi elemen teratas dalam tumpukan yang dimaksud. Sebagai contoh, jika T.Atas = 5, berarti dalam tumpukan ada 5 elemen, yaitu T.Isi [1],…, T.Isi [5]. Jika ada data yang diambil, maka nilai medan T.Atas dikurangi 1 menjadi 4, yang berarti T.Isi[4] adalah elemen teratas. Sebaliknya, jika kedalam tumpukan ditambahkan sebuah elemen, maka nilai T.Atas ditambah dengan 1 menjadi 6, sehingga T.Isi[6] adalah elemen teratas.

3.2 Operasi Pada Tumpukan

Ada dua operasi dasar yang dapat dilaksanakan pada sebuah tumpukan, yaitu operasi menyisipkan data, atau mem*push* data, dan operasi menghapus data atau mem*pop* data. Karena ke dalam tumpukan dapat di-*push* data, maka tumpukan juga sering disebut dengan *pushdown list.*

1. Operasi Push

Sekarang marilah kita menyusun prosedur untuk operasi *push*. Dengan menyajikan tumpukan seperti di atas, maka operasi *push* dengan mudah dapat diimplementasikan, yaitu:

```
procedure PUSH (var T : Tumpukan; X : integer);
begin
   T.Atas := T.Atas + 1;
   T.Isi[T.Atas] := X
end;
```

Prosedur di atas akan menyiapkan tempat untuk X yang akan di *push* ke dalam tumpukan, yaitu dengan menambah nilai medan T.Atas dengan 1 dan kemudian menyisipkan X ke dalam larik T.Isi. Prosedur di atas nampaknya sudah mencukupi. Tetapi jika pada suatu saat nilai T.Atas sama dengan MaxElemen dan akan di-*push* lagi, maka akan terjadi *overflow* pada larik T.Isi berhubung deklarasi banyaknya elemen larik tersebut tidak mencukupi. Dengan demikian, pada prosedur di atas perlu ditambah dengan *testing* untuk memastikan bahwa sebelum suatu data di-*push* nilai T.Atas belum mencapai MaxElemen. Dengan demikian prosedur di atas perlu diubah menjadi:

```
procedure PUSH (var T : Tumpukan; X : integer);
begin
        if T.Atas = MaxElemen then
                writeln ('Tumpukan Sudah Penuh')
        else
        begin
                T.Atas := T.Atas + 1;
                T.Isi[T.Atas] := X
        end
end;
```

Dengan ditambahkannya *testing* untuk mencek nilai T.Atas, maka prosedur di atas menjadi lebih sempurna. Kita juga dapat mengubah pesan yang ditampilkan menggunakan statemen ***writeln*** dengan data lain yang bertipe ***boolean***, sehingga pesan salahnya diletakkan pada program utama. Dengan menggunakan kontrol berupa data bertipe **boolean**, maka prosedur di atas dapat diubah menjadi seperti terlihat pada prosedur 1.

```
procedure PUSH (var T: Tumpukan; var Penuh: boolean;
                X: integer);
begin
  if T.Atas = MaxElemen then
        {* Tumpukan Sudah Penuh *}
        penuh := true
  else
        begin`
                Penuh := false;
```

```
            T.Atas := inc(T.Atas);
            T.Isi[T.Atas] := X
        end
end;
```

Pemanggilan prosedur di atas dari program utama atau dari bagian lain dilaksanakan dengan:

PUSH (T, Penuh, X);

dengan T adalah perubah bertipe Tumpukan, Penuh bertipe ***boolean*** dan x bertipe **integer** (dalam contoh ini). Berdasarkan nilai perubah penuh dapat ditentukan langkah selanjutnya yang perlu diambil.

2. Operasi Pop

Operasi pop adalah operasi untuk menghapus elemen yang terletak pada posisi paling atas dari sebuah tumpukan. Sama halnya dengan operasi push, maka dengan deklarasi tumpukan seperti di atas, prosedur untuk operasi pop dapat dengan mudah kita implementasikan, yaitu:

```
procedure POP (var T : Tumpukan);
begin
   T.Atas := T.Atas – 1
end;
```

Prosedur di atas nampaknya sudah mencukupi untuk mempop elemen tumpukan. Timbul pertanyaan, seandainya tumpukan sudah kosong lalu apa yang akan dipop? Jawabnya tentu saja adalah tidak mungkin mempop suatu tumpukan jika tumpukan tersebut sudah kosong. Dengan demikian, prosedur di atas perlu ditambah *testing* untuk mencek kosong tidaknya sebuah tumpukan sebelum proses mempop suatu elemen tumpukan dilaksanakan. Tumpukan yang kosong ditunjukkan dengan nilai T.Atas sama dengan 0. Sehingga dapat dilengkapi prosedur di atas menjadi:

```
procedure POP (var T : Tumpukan);
begin
   if T.Atas = 0 then
        writeln ('Tumpukan Sudah Kosong')
   else
        T.Atas := T.Atas – 1
end;
```

Cara lain untuk melihat kosong tidaknya tumpukan adalah dengan membuat suatu fungsi yang menghasilkan suatu data yang bertipe **boolean**. Cara ini lebih disarankan dari cara di atas, karena dengan mengetahui nilai fungsi tersebut kita segera dapat tahu kosong tidaknya suatu tumpukan. Alasan lain adalah supaya sifat modularitas program tetap dipertahankan. Kecuali itu, seringkali juga diperlukan mengetahui nilai data yang baru saja dipop. Untuk itu prosedur di atas dapat diubah menjadi sebuah fungsi seperti tersaji dalam Prosedur 2. (lengkap dengan fungsi untuk mentest kosong tidaknya tumpukan).

Dalam program utama, kita dapat memanggil fungsi di atas dengan:

X := POP (T);

dengan X (dalam contoh ini) adalah data bertipe ***integer***.

3.3 Contoh Pemakaian Tumpukan

Untuk lebih memahami operasi yang terjadi pada tumpukan, berikut disajikan contoh program yang memanfaatkan tumpukan untuk membalik kalimat. Dalam hal ini yang dibalik adalah seluruh kalimat, bukan per kata. Anda dapat mencoba, dengan mengacu pada program ini, membalik kalimat dengan melakukan pembalikan perkata. Sebagai contoh, jika kalimat yang dibaca adalah:

BELAJAR PASCAL ADALAH MUDAH DAN MENYENANGKAN

setelah dibalik, maka kalimat di atas dapat menjadi:

NAKGNANEYNEM NAD HADUM HALADA LACSAP RAJALEB

```
{Fungsi untuk mem-pop elemen dari tumpukan. Fungsi ini berisi fungsi lain
untuk mengetahui kosong tidaknya tumpukan yang elemennya akan di-pop}
Function pop (var T : tumpukan) : integer;

{fungsi untuk mengetahi kosong tidaknya tumpukan yang elemennya akan di-
pop}
function kosong (var T : tumpukan) : boolean;
begin
        kosong := (T.Atas = 0)
end;       { fungsi kosong }
{ ********************** }
{        fungsi pop              }
{ ********************** }
begin
        if kosong (T) then
                pop := 0
        else
                begin
                        pop := T.Isi[T.Atas];
                        T.Atas := dec[T.Atas];
                End;
End;     { Fungsi POP }
```

Dalam program yang akan disajikan, kalimat yang akan dibalik disimpan dalam suatu perubah. Kemudian dengan menggunakan proses tumpukan, setiap karakter diambil dan dimasukkan dalamnya. Dengan cara ini, karakter pertama dari kalimat tersebut akan menempati bagian bahwa tumpukan dan karakter terakhir akan

menempati bagian atas tumpukan. Dengan demikian, jika semua karakter dalam tumpukan di pop, kalimat akan terbalik.

Program selengkapnya tersaji di bawah ini.

```
program balik_kalimat;
uses
   crt;
const
    elemen = 255;    { batas maksimum karakter }
type
   s255 = string[elemen];
   tumpukan = record
                isi : s255;
                atas : 0..elemen
            end;
var
  t : tumpukan;    { nama tumpukan }
  i : integer;     { pencacah }
  kalimat : s255;  { kalimat yang dibalik }
  n,m : char;
{ ****************************** }
{ prosedur inisialisasi tumpukan }
{ ****************************** }
procedure awalan ( var t : tumpukan );
begin
    t.atas := 0;
end;           { akhir prosedur awalan }
{ ********************************************************** }
{ prosedur untuk memastikan elemen ke dalam tumpukan       }
{ dalam hal ini cacah karakter maksimum tidak boleh dari 255 }
{ ********************************************************** }
procedure push ( var t : tumpukan; x : char );
begin
    t.atas := t.atas + 1;
    t.isi[t.atas] := x;
end;         { akhir Prosedur push }
{ ****************************************** }
{ fungsi untuk mengambil elemen dari tumpukan }
{ ****************************************** }
function pop ( var t : tumpukan ) : char;
begin
    pop := t.isi[t.atas];
    t.atas := t.atas - 1;
end;    { akhir fungsi pop }

{ ************************ }
{      program utama       }
{ ************************ }
```

```
begin
    clrscr;
    awalan (t);
    writeln (' Tumpukan untuk membalik kalimat  ');
    writeln (' ------------------------------- ');
    writeln;
    { kalimat yang akan dibalik }
    writeln ('Isikan sembarang kalimat : ');
    readln (kalimat);
    writeln;
    writeln(' Kalimat asli : '); ;writeln (kalimat);
    writeln('setelah dibalik : ');
    { mempush kalimat ke dalam tumpukan }
    for i := 1 to length(kalimat) do
    push(t, kalimat[i]);
    { mempop isi tumpukan sehingga diperoleh kalimat }
    { yang dibaca terbalik pembacaannya              }
    for i := 1 to length(kalimat) do
    write (pop(t));
    readln;
end.   { akhir program utama }
```

****

ouput program tersebut tersaji sebagai berikut:

****

```
Tumpukan untuk membalik kalimat
 ----------------------------------------:
Isikan sembarang kalimat :
saya mau pergi ke pasar
Kalimat asli :
saya mau pergi ke pasar
setelah dibalik :
rasap ek igrep uam ayas
----------------------------------------
```

Contoh program tumpukan sebagai berikut:

```
program balik_kalimat;
uses
   crt;
const
    elemen = 255;     { batas maksimum karakter }
type
   s255 = string[elemen];
   tumpukan = record
                isi : s255;
                atas : 0..elemen
            end;
var
  t : tumpukan;     { nama tumpukan }
```

```
  i,nn,th : integer;      { pencacah }
  kalimat : s255;   { kalimat yang dibalik }
  n,m,lagi : char;
{ ***************************** }
{ prosedur inisialisasi tumpukan }
{ ***************************** }
procedure awalan ( var t : tumpukan );
begin
    t.atas := 0;
end;            { akhir prosedur awalan }
{ ************************************************************ }
{ prosedur untuk memastikan elemen ke dalam tumpukan         }
{ dalam hal ini cacah karakter maksimum tidak boleh dari 255 }
{ ************************************************************ }
procedure push ( var t : tumpukan; x : char );
begin
    t.atas := t.atas + 1;
    t.isi[t.atas] := x;
end;          { akhir Prosedur push }
{ ***************************************** }
{ fungsi untuk mengambil elemen dari tumpukan }
{ ***************************************** }
function pop ( var t : tumpukan ) : char;
begin
    pop := t.isi[t.atas];
    t.atas := t.atas - 1;
end;    { akhir fungsi pop }
{ ************************* }
{       program utama       }
{ ************************* }
begin
    clrscr;
    awalan (t);
    writeln (' Tumpukan untuk membalik kalimat  ');
    writeln (' ------------------------------ ');
    writeln;
    { kalimat yang akan dibalik }
    writeln ('Isikan sembarang kalimat : ');
    readln (kalimat);
    writeln;
    nn:=length(kalimat);
    { mempush kalimat ke dalam tumpukan }
    for i := 1 to nn do
    push(t, kalimat[i]);
    { mempop isi tumpukan sehingga diperoleh kalimat }
    { yang dibaca terbalik pembacaannya              }
    th:=0;
    repeat
```

```
        th:=th+1;
        writeln;
        write('akan tarik karakter ? ( Y/T): ');readln(lagi);
        if  ('Y'= upcase(lagi)) then
        begin
            write('karakter yang ditarik adalah karakter paling akhir --> ');
            writeln (pop(t));
            readln;
        end
    until  (('Y'<> upcase(lagi)) or (th=nn) );
         if not (('Y'<> upcase(lagi)) or (th=nn)) then  write(pop(t));
end.   { akhir program utama }
```

Hasil program tersebut sebgai berikut:
Tumpukan untuk membalik kalimat

```
Isikan sembarang kalimat :
pasar

akan tarik karakter ? ( Y/T): y
karakter yang ditarik adalah karakter paling akhir --> r

akan tarik karakter ? ( Y/T): y
karakter yang ditarik adalah karakter paling akhir --> a

akan tarik karakter ? ( Y/T): y
karakter yang ditarik adalah karakter paling akhir --> s

akan tarik karakter ? ( Y/T): y
karakter yang ditarik adalah karakter paling akhir --> a

akan tarik karakter ? ( Y/T): y
karakter yang ditarik adalah karakter paling akhir --> p
```

BAB 4
ANTRIAN (QUEUE)

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi antrian, implementasi antrian dengan larik dan fungsi pada antrian

Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi antrian, implementasi antrian dengan larik dan fungsi pada antrian

PENYAJIAN

4.1 Pengertian Antrian

Antrian adalah suatu kumpulan data yang mana penambahan elemen hanya bisa dilakukan pada suatu ujung (disebut dengan sisi belakang atau *rear*), dan penghapusan (pengambilan elemen) dilakukan lewat ujung lain (disebut dengan sisi depan atau *front*). Seperti kita ketahui, tumpukan menggunakan prinsip "masuk terakhir keluar pertama" atau LIFO (*Last In First Out*), maka pada antrian prinsip yang digunakan adalah "masuk pertama keluar pertama" atau FIFO (*First In First Out*). Dengan kata lain, urutan keluar elemen akan sama dengan urutan masuknya.

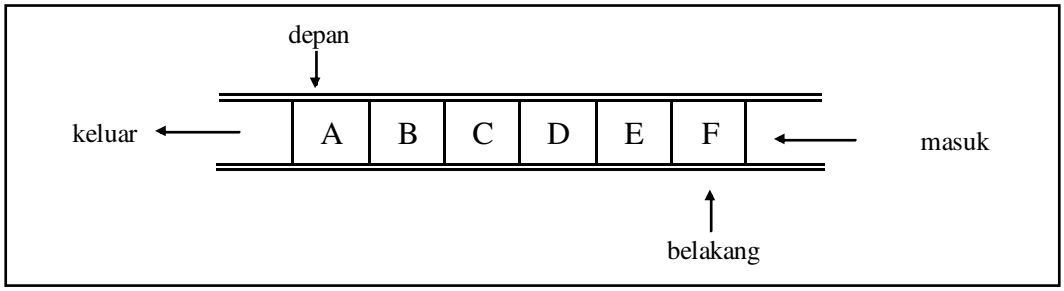
Antrian banyak kita jumpai dalam kehidupan sehari-hari. Mobil-mobil yang antri membeli karcis di pintu jalan tol akan membentuk antrian; orang-orang yang membeli karcis untuk menyaksikan film akan membentuk antrian; para nasabah bank yang melakukan transaksi (mengambil dan menabung) membentuk antrian, dan contoh-contoh lain yang banyak kita jumpai dalam kehidupan sehari-hari.

Contoh lain yang lebih relevan dalam bidang komputer adalah pemakaian sistem komputer berbagi waktu (*time-sharing computer system*) dimana ada sejumlah pemakai yang akan menggunakan sistem tersebut secara serempak. Karena sistem ini biasanya menggunakan sebuah prosesor dan sebuah pengingat utama, maka jika prosesor sedang dipakai oleh seorang pemakai, pemakai-pemakai lain harus antri

sampai gilirannya tiba. Antrian ini mungkin tidak akan dilayani secara FIFO murni, tetapi didasarkan pada suatu prioritas tertentu. Antrian yang mengandung unsur prioritas dinamakan dengan antrian berprioritas (*priority queue*) yang juga akan dijelaskan dalam kuliah ini.

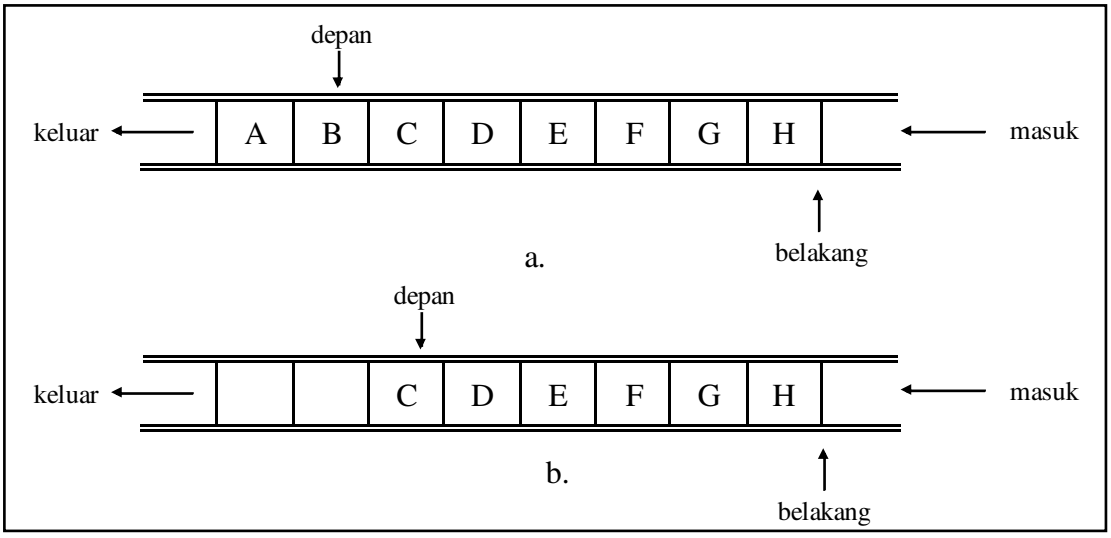
4.2 Implementasi Antrian Dengan Larik

Di atas telah dijelaskan bahwa antrian sebenarnya juga merupakan satu kumpulan data. Dengan demikian tipe data yang sesuai untuk menyajikan antrian adalah menggunakan larik dan senarai berantai. Penyajian antrian menggunakan senarai berantai juga akan dijelaskan pada kuliah ini.



Gambar 4.1 Contoh antrian dengan 6 elemen

Gambar 4.1. di atas menunjukkan contoh penyajian antrian menggunakan larik. Antrian di atas berisi 6 elemen, yaitu A, B, C, D, E, dan F. Elemen A terletak di bagian depan antrian dan elemen F terletak di bagian belakang antrian. Dengan demikian, jika ada elemen baru yang akan masuk, maka ia akan diletakkan di sebelah kanan F (pada gambar di atas). Jika ada elemen yang akan dihapus, maka A akan dihapus lebih dahulu. Gambar 2.a. menunjukkan antrian setelah berturut-turut dimasukkan G dan H. Gambar 2.b. menunjukkan antrian Gambar 4.2.a. setelah elemen A dan B dihapus.



Gambar 4.2. Contoh penambahan dan penghapusan elemen pada suatu antrian

Seperti halnya pada tumpukan, maka dalam antrian kita juga mengenal adanya dua operasi dasar, yaitu menambah elemen baru yang akan kita tempatkan di bagian belakang antrian dan menghapus elemen yang terletak di bagian depan antrian. Di samping itu seringkali kita juga perlu melihat apakah antrian mempunyai isi atau dalam keadaan kosong.

Operasi penambahan elemen baru selalu bisa kita lakukan karena tidak ada pembatasan banyaknya elemen dari suatu antrian. Tetapi untuk menghapus elemen, maka kita harus melihat apakah antrian dalam keadaan kosong atau tidak. Tentu saja kita tidak mungkin menghapus elemen dari suatu antrian yang sudah kosong.

Untuk menyajikan antrian menggunakan larik, maka kita membutuhkan deklarasi antrian, misalnya, sebagai berikut:

```
Const Max_Elemen = 100;
type Antri = array[1..Max_Elemen] of integer;
var     Antrian : Antri;
        Depan,
        Belakang : integer;
```

Dalam deklarasi di atas, elemen antrian dinyatakan dalam tipe integer. Perubah **Depan** menunjukkan posisi elemen pertama dalam larik; perubah **Belakang** menunjukkan posisi elemen terakhir dalam larik. Dengan menggunakan larik, maka kejadian *overflow* sangat mungkin, yakni jika antrian telah penuh, sementara kita masih ingin menambah terus. Dengan mengabaikan kemungkinan adanya *overflow,* maka penambahan elemen baru, yang dinyatakan oleh perubah x, bisa kita implementasikan dengan statemen:

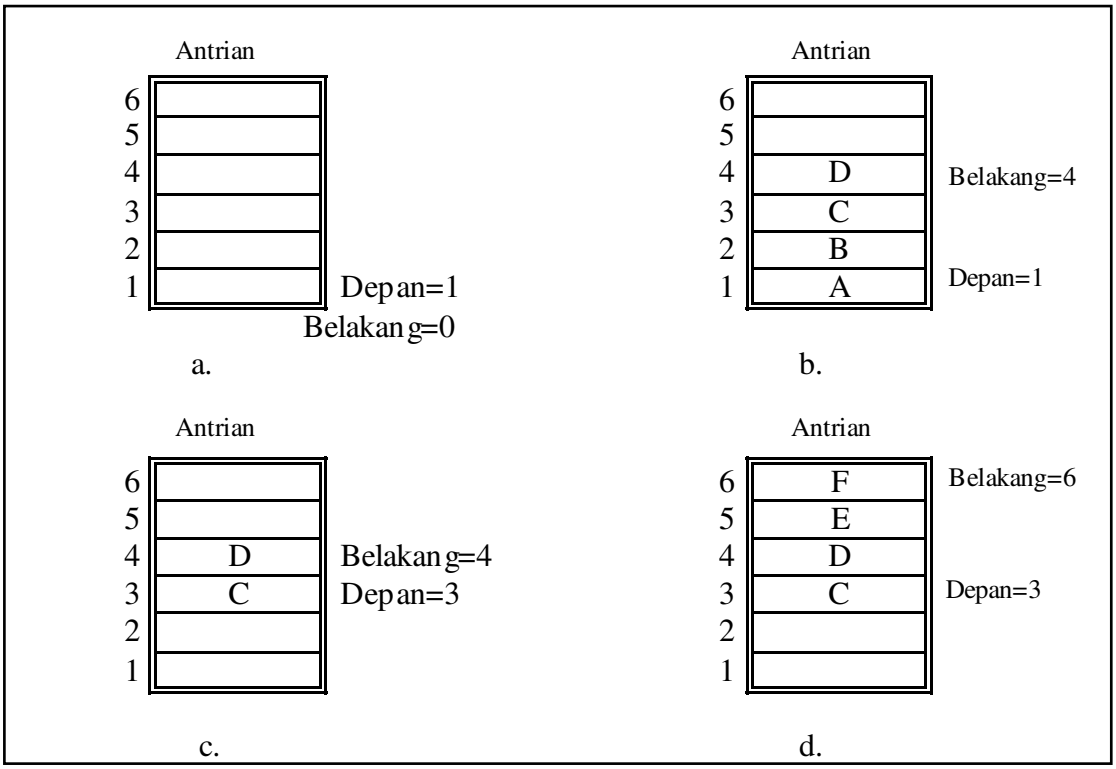
```
Belakang := Belakang + 1;
Antrian[Belakang] := X;
```

Operasi penghapusannya bisa diimplementasikan dengan:

```
X := Antrian[Depan];
Depan := Depan + 1;
```

Pada saat permulaan, **Belakang** dibuat sama dengan 0 dan **Depan** dibuat sama dengan 1, dan antrian dikatakan kosong jika **Belakang < Depan**. Banyaknya elemen yang ada dalam antrian dinyatakan sebagai **Belakang** – **Depan** + 1. Sekarang marilah kita tinjau implementasi menambah dan menghapus elemen seperti diperlihatkan di atas. Gambar 4.3 menunjukkan larik dengan 6 elemen untuk menyajikan sebuah antrian (dalam hal ini Max_Elemen = 6). Pada saat permulaan (Gambar 4.3.a.), antrian dalam keadaan kosong. Pada Gambar 4.3.b. terdapat 4 buah elemen yang telah ditambahkan. Dalam hal ini nilai **Depan** = 1 dan **Belakang** = 4.

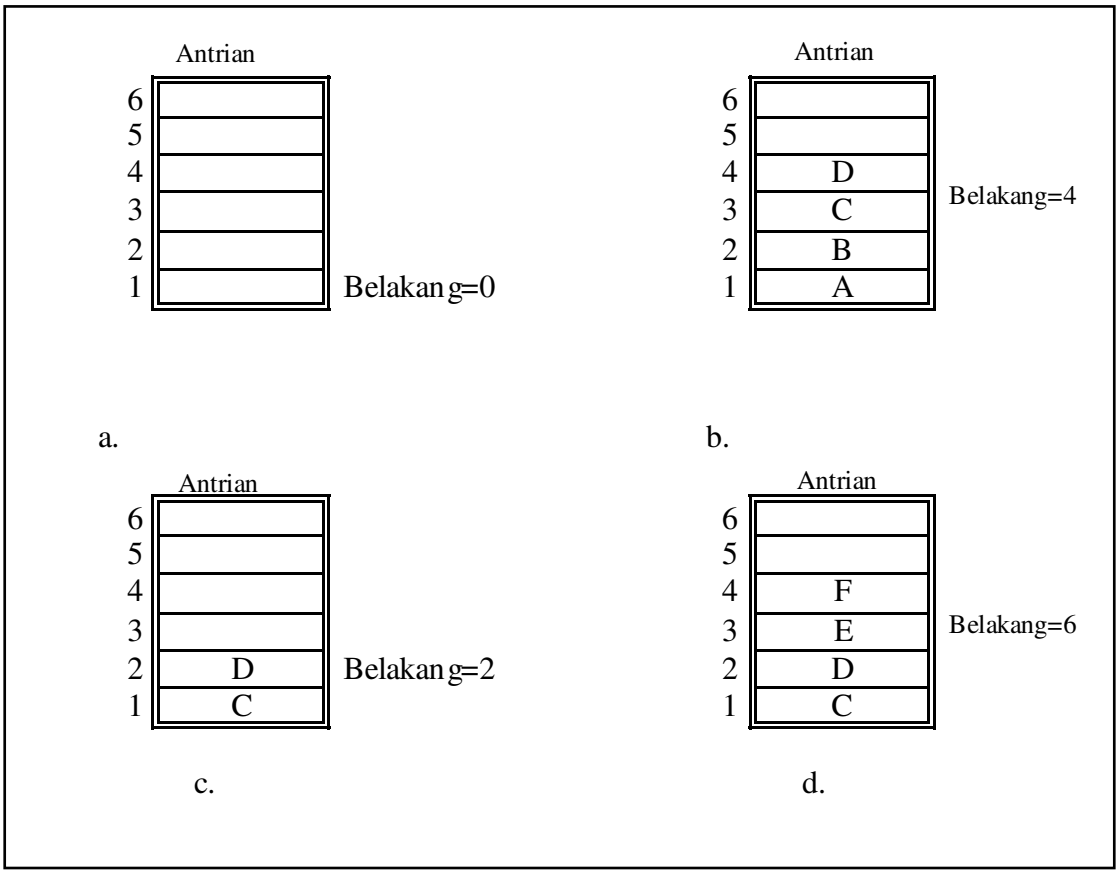
Gambar 4.3.c. menunjukkan antrian setelah dua elemen dihapus. Gambar 4.3.d. menunjukkan antrian setelah dua elemen baru ditambahkan. Banyaknya elemen dalam antrian adalah 6 – 3 + 1 = 4 elemen. Karena larik terdiri dari 6 elemen, maka sebenarnya kita masih bisa menambah eleman lagi. Tetapi, jika kita ingin menambah elemen baru, maka nilai **Belakang** harus ditambah satu, menjadi 7. Padahal larik Antrian hanya terdiri dari 6 elemen, sehingga tidak mungkin ditambah lagi, meskipun sebenarnya larik tersebut masih kosong di dua tempat. Bahkan dapat terjadi suatu situasi dimana sebenarnya antriannya dalam keadaan kosong, tetapi tidak ada elemen yang bisa tambahkan kepadanya. Dengan demikian, penyajian di atas tidak dapat diterima.



Gambar 4.3. Ilustrasi penambahan dan penghapusan elemen pada sebuah antrian

Salah satu penyelesaiannya adalah dengan mengubah prosedur untuk menghapus elemen, sedemikian rupa sehingga jika ada elemen yang dihapus, maka semua elemen lain digeser sehingga antrian selalu dimulai dari **Depan** = 1. Dengan cara ini, maka sebenarnya perubah **Depan** ini tidak diperlukan lagi hanya perubah **Belakang** saja yang diperlukan, karena nilai Depan selalu sama dengan 1. Berikut adalah rutin penggeserannya (dengan mengabaikan kemungkinan adanya *underflow*).

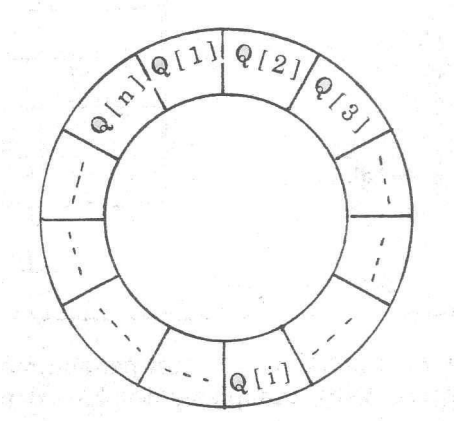
```
X := Antrian[1];
for I := 1 to Belakang – 1 do
        Antrian[I] := Antrian[I+1];
Belakang := Belakang – 1;
```



Gambar 4.4. Ilustrasi penambahan dan penghapusan elemen pada sebuah antrian dengan penggeseran

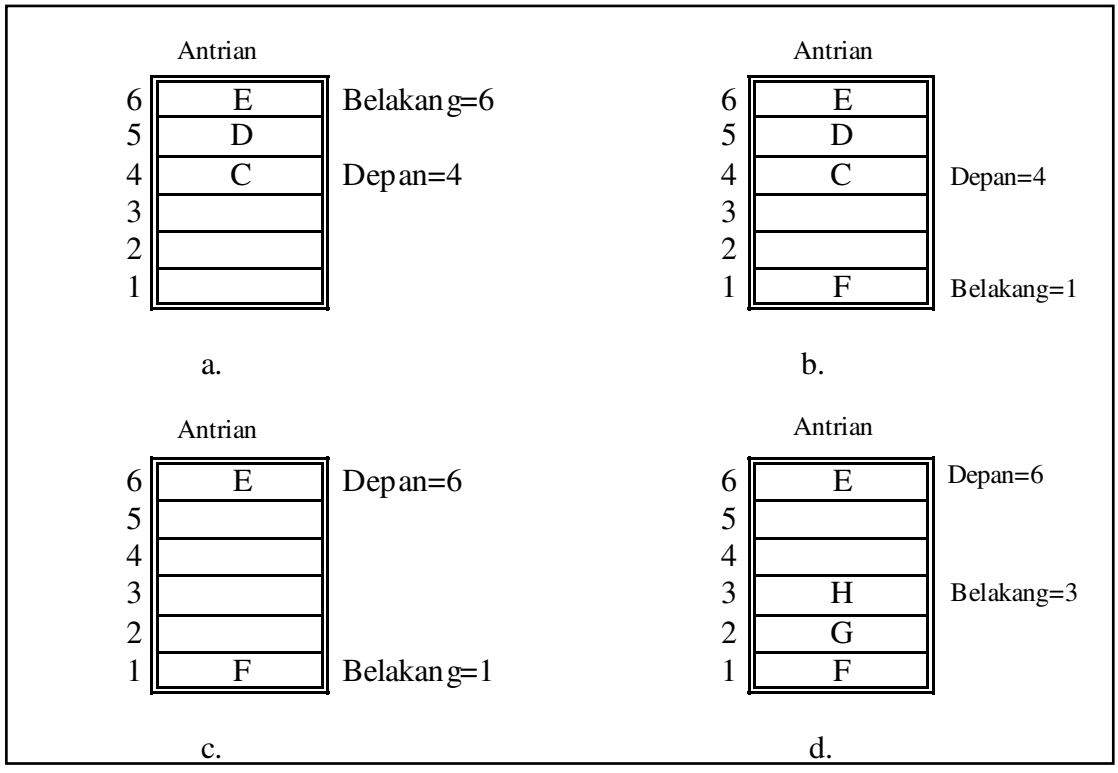
Dalam hal ini antrian kosong dinyatakan sebagai nilai **Belakang** = 0. Gambar 4.4. menunjukkan ilustrasi penambahan dan penghapusan elemen pada sebuah antrian dengan penggeseran elemen. Cara ini kelihatannya sudah memecahkan persoalan yang kita miliki. Tetapi jika kita tinjau lebih lanjut, maka ada sesuatu yang harus dibayar lebih mahal, yaitu tentang penggeseran elemen itu sendiri. Jika kita mempunyai antrian dengan 10000 elemen, maka waktu terbanyak yang dihabiskan sesungguhnya hanya untuk melakukan proses penggeseran. Hal ini tentu saja sangat tidak efisien. Dengan melihat gambar-gambar ilustrasi, proses penambahan dan penghapusan elemen antrian sebenarnya hanya mengoperasikan sebuah elemen, sehingga tidak perlu ditambah dengan sejumlah operasi lain yang justru menyebabkan tingkat ketidak-efisienan bertambah besar.

Pemecahan lain adalah dengan memperlakukan larik yang menyimpan elemen antrian sebagai larik yang memutar (*circular*), bukan lurus (*straight*), yakni dengan membuat elemen pertama larik terletak berdekatan langsung dengan elemen terakhir larik. Dengan cara ini, meskipun elemen terakhir telah terpakai, elemen baru masih tetap bisa ditambahkan pada elemen pertama, sejauh elemen pertama dalam keadaan kosong. Gambar berikut ini menunjukkan ilustrasi antrian Q yang diimplementasikan dengan larik yang memutar.



Gambar 4.5. Implementasi antrian dengan larik yang memutar

Marilah kita perhatikan contoh berikut (dalam contoh ini, larik kita gambarkan mendatar untuk lebih mempermudah pemahamannya). Gambar 4.6a. menunjukkan antrian telah terisi dengan 3 elemen pada posisi ke 4, 5 dan 6 (dalam hal ini Max_Elemen = 6).



Gambar 4.6. Ilustrasi penambahan dan penghapusan elemen pada sebuah antrian yang diimplementasikan dengan larik yang memutar

Gambar 4.6.b. menunjukkan antrian setelah ditambah dengan sebuah elemen baru, F. Bisa anda perhatikan, bahwa karena larik hanya berisi 6 elemen, sedangkan pada Gambar 4.6.a. posisi ke 6 telah diisi, maka elemen baru akan menempati posisi

ke 1. Gambar 4.6.c. menunjukkan ada 2 elemen yang dihapus dari antrian, dan Gambar 4.6.d. menunjukkan ada 2 elemen baru yang ditambahkan ke antrian. Bagaimana jika elemen antrian tersebut akan dihapus lagi? Diserahkan pada Anda sekalian untuk meneruskan pelacakan di atas.

Dengan melihat pada ilustrasi di atas, kita bisa menyusun prosedur untuk menambah elemen baru ke dalam antrian. Prosedur di bawah ini didasarkan pada deklarasi berikut:

```
Const Max_Elemen = 100;
type  Antri = array[1..Max_Elemen] of char;
var   Antrian          : Antri;
      Depan, Belakang : integer;
```

Untuk awalan, maka kita tentukan nilai perubah Depan dan Belakang sebagai:

```
Depan : = 0;
Belakang : = 0;
```

Prosedur selengkapnya untuk menambahkan elemen baru adalah sebagai berikut:

```
{*********************************************************
*Program 6.1. (PROG6-1.PAS                                *
* Prosedur menambah elemen baru pada antrian.             *
* Q adalah nama antrian dan X adalah elemen               *
* baru yang ditambahkan                                   *
*********************************************************}
procedure TAMBAH (var Q : Antri; X : char);
begin
      if Belakang := Max_Elemen then Belakang := 1
      else Belakang := Belakang + 1;
      if Depan = Belakang then
            writeln('ANTRIAN SUDAH PENUH')
      else Q [Belakang] := X
end;
```

Program 1. Prosedur untuk menambah elemen baru pada sebuah antrian

Untuk menghapus elemen, terlebih dahulu kita harus melihat apakah antrian dalam keadaan kosong atau tidak. Berikut disajikan satu fungsi untuk mencek keadaan antrian.

```
{*************************************
 * Fungsi untuk mencek keadaan antrian *
 *************************************}
 function KOSONG(Q :Antri) : boolean;

begin
  KOSONG := (Depan = Belakang);
end;
```

Program 2. Fungsi untuk mencek keadaan antrian

Dengan memperhatikan fungsi di atas, maka kita bisa menyusun fungsi lain untuk menghapus elemen, yaitu:

```
{*********************************************
 * Fungsi untuk menghapus elemen dari antrian *
 *********************************************}
 function HAPUS(var Q :Antri) : char;

begin
  if KOSONG(Q)then
    writeln('ANTRIAN KOSONG…')
  else
    begin
      HAPUS := Q[Depan];
      If Depan := Max_Elemen then
        Depan := 1
      else
        Depan := Depan + 1;
    end;
end;
```

Program 3. Fungsi untuk menghapus elemen dari antrian

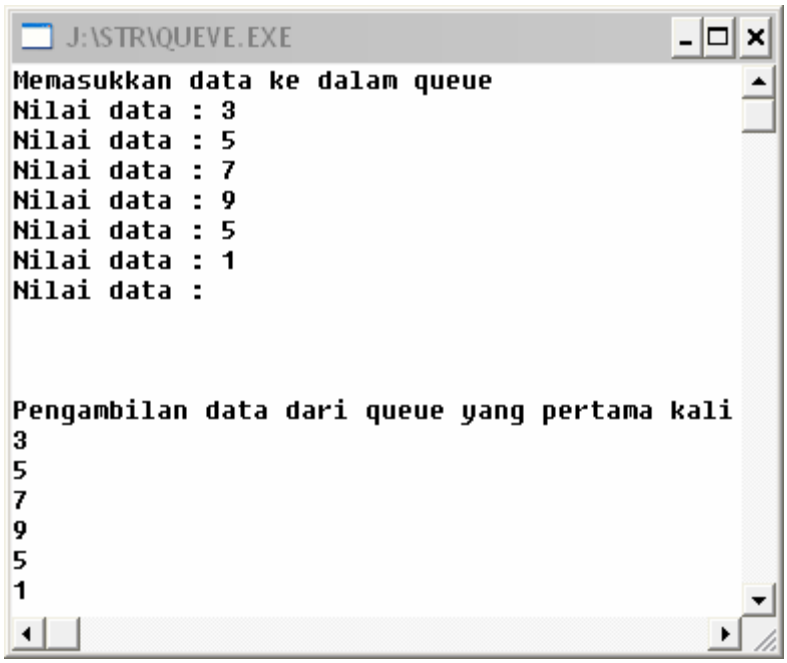
Dengan memperhatikan program-program di atas, bisa kita lihat bahwa sebenarnya elemen yang terletak di depan antrian, menempati posisi (subskrip) ke Depan + 1 pada larik yang digunakan. Sehingga sesungguhnya, banyaknya elemen maksimum yang bisa disimpan adalah sebesar Max_Elemen – 1, kecuali pada saat permulaan.

4.3 Contoh program antrian

```
Program contoh_queue;
uses wincrt;

Type
        list=^node;
                node=record
                isi:char;
                next:List;
        end;
        Queue=record
                depan,belakang:List;
        end;
{-----------------------------------------------------------------}
Procedure initQueue(var q:queue);
begin
        q.depan:=nil; q.belakang:=nil;
end;
```

```
{------------------------------------------------------------------}
Procedure EnQueue(data:char; var Q:queue);
var             b:list;
begin
        new(b); b^.isi:=data; b^.next:=nil;
        if q.belakang=nil then begin
                q.belakang:=b;q.depan:=b;
        end else begin
                q.belakang^.next:=b; q.belakang:=b;
        end;
end;
{------------------------------------------------------------------}
Procedure Dequeue(var q:queue; var hasil:char);
var b:list;
begin
        if q.depan <> nil then begin
        hasil:=q.depan^.isi; b:=q.depan;
        q.depan:=b^.next; dispose(b);
        if q.depan=nil then q.belakang:=nil;
        end;
end;
{------------------------------------------------------------------}
var  x:char;
        q:queue;
        begin
                clrscr;
                initqueue(q);
                writeln('Memasukkan data ke dalam queue');
                repeat
                        write('Nilai data : ');x:=upcase(readkey);writeln(x);
                        if x<>#13 then EnQueue(x,Q);
                until x=#13;

                writeln;
                readln;
                writeln('Pengambilan data dari queue yang pertama kali');

                while q.depan<>nil do
                        begin
                                deQueue(Q,x);writeln(x);
                        end;
                readln;
                writeln('Pengambilan data dari queue yang kedua kali');
                while q.depan<>nil do
                        begin
                                deQueue(Q,x);writeln(x);
                        end;
end.
```

Hasil eksekusi program antrian



Gambar 4.7 Hasil eksekusi program antrian

BAB 5
SENARAI BERANTAI (LINKED LIST)

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi linked list, operasi pada linked list, penyajian linked list dan disertai dengan contoh program linked list.

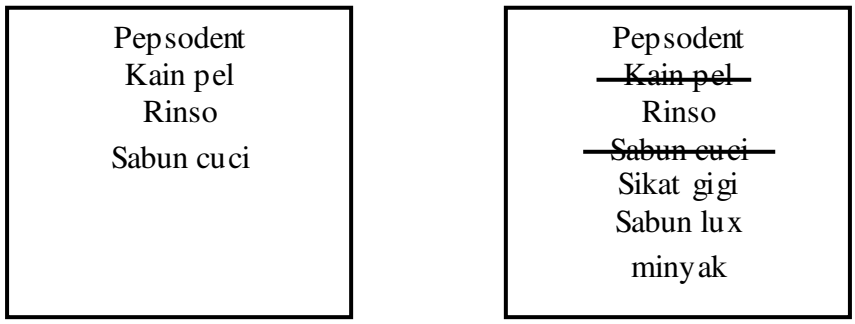
Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi linked list, operasi pada linked list, penyajian linked list dan dapat membuat program linked list

PENYAJIAN

5.1 Pengertian linked list

Dalam pemakaian sehari-hari istilah senarai (*List*) adalah kumpulan linear sejumlah data. Gambar 5.1.a. dibawah ini menunjukkan senarai yang berisi daftar belanja, yang berupa barang pertama, kedua, ketiga dan seterusnya. Untuk hari berikutnya, maka daftar tersebut bisa berubah sesuai dengan barang yang harus dibeli lagi atau barang yang tidak perlu dibeli lagi. Gambar 5.1.b. menunjukkan daftar belanjaan semula setelah ditambah dengan 3 barang lain dan menghapus 2 barang (dengan mencoret) yang tidak perlu dibeli lagi.



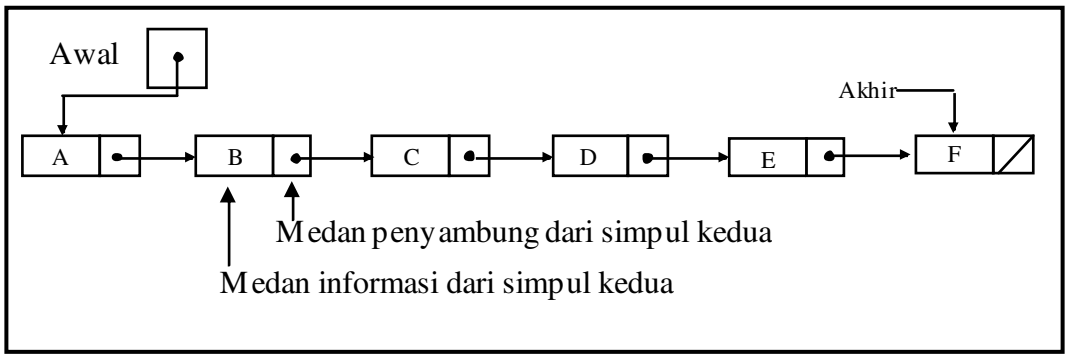
a. b.

Gambar 5.1. Daftar barang belanjaan.

Pengolahan data yang kita lakukan menggunakan komputer seringkali mirip dengan ilustrasi di atas, yang antara lain berupa penyimpanan data dan pengolahan lain dari sekelompok data yang telah terorganisir dalam sebuah urutan tertentu. Salah satu cara untuk menyimpan sekumpulan data yang kita miliki adalah menggunakan **larik**. Keuntungan dan kekurangan pemakaian larik untuk menyimpan sekelompak data yang banyaknya selalu berubah dan tidak diketahui dengan pasti kapan penambahan atau penghapusan akan berakir. Cara lain untuk menyimpan data, juga dapat digunakan dengan bantuan data yang bertipe **pointer**. Dalam hal ini masing-masing data dengan sebuah medan yang bertipe **pointer** perlu digunakan.

5.1.1 Senarai Berantai

Dengan menggunakan data bertipe **pointer** yang akan dibahas pada kuliah berikut disinggung bahwa penggunaan **pointer** sangat mendukung dalam pembentukan struktur data dinamis. Salah satu struktur data dinamis yang paling sederhana adalah senarai berantai (*linked list*). Dengan demikian, senarai berantai adalah kumpulan komponen yang disusun secara berurutan dengan bantuan **pointer**. Masing-masing komponen dinamakan dengan **simpul** (*node*). Dengan demikian, setiap **simpul** dalam suatu senarai berantai terbagai menjadi dua bagian. Bagian pertama, disebut medan informsi, berisi informasi yang akan disimpan dan diolah. Bagian kedua, disebut dengan penyambung (*link field*), berisi alamat simpul berikutnya. Gambar 5.2 di bawah ini menunjukan diagram skematis dari senarai berantai dengan 6 simpul. Setiap simpul digambarkan dalam dua bagian. Bagian kiri adalah medan informasi. Bagian kanan berupa medan penyambung, sehingga dalam diagram digambarkan sebagai anak panah. Perlu diingat bahwa medan penyambung sebenarnya adalah suatu **pointer** yang menunjuk ke simpul berikutnya, sehingga nilai dari medan ini adalah alamat suatu lokasi tertentu dalam memori. Dari Gambar 5.2. **Pointer Awal**, yang bukan merupakan bagian dari simpul senarai, menunjuk ke simpul pertama dari senarai tersebut. Medan penyambung (**pointer**) dari suatu simpul yang tidak menunjuk simpul lain disebut dengan **pointer kosong**, yang nilainya dinyatakan sebagai nil (nil adalah kata baku Pascal yang berarti bahwa **pointer** tidak menunjuk ke suatu simpul, dan nilainya sama dengan 0 atau bilangan negetif). Dengan memperhatikan Gambar 5.2. Kita bisa melihat bahwa dengan hanya sebuah **pointer Awal** saja maka kita bisa membaca semua informasi yang tersimpan dalam senarai.



Gambar 5.2. Contoh senarai berantai dengan 6 simpul.

Senarai berantai sangat berguna dalam terapan-terapan sehari-hari dimana komponen-komponen dalam jumlah sangat besar harus dijaga supaya selalu dalam keadaan terurutkan menurut kriteria tertentu, misalnya secara alphabetis, dan dimana komponen-komponennya sering dihapus atau ditambahkan.

5.2 Operasi Pada Senarai Berantai

Ada sejumlah operasi yang bisa kita lakukan pada sebuah senarai berantai, yaitu operasi membaca isi senarai (*traversal*), menambah simpul, menghapus simpul dan mencari informasi pada suatu senarai berantai. Berikut akan dijelaskan satu persatu.

1. Menambah Simpul

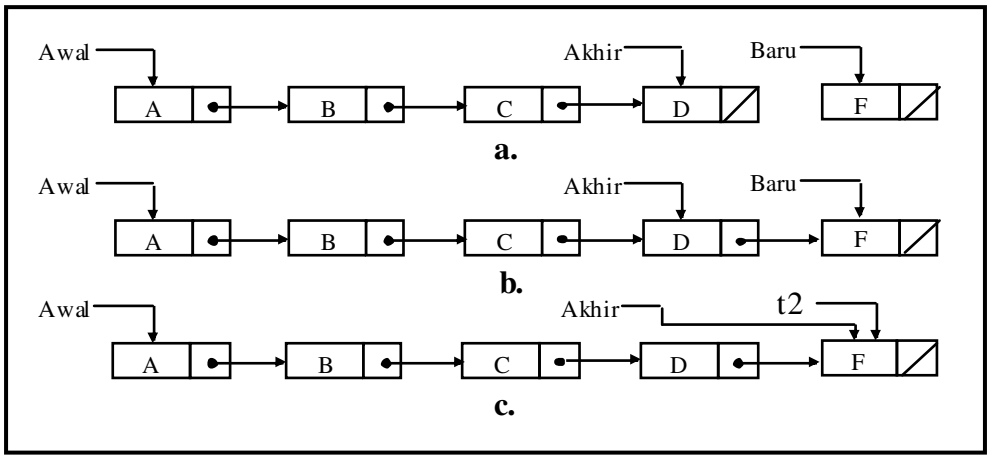
Operasi pada senarai berantai yang akan dijelaskan pertama kali adalah operasi untuk **menambah simpul**. Operasi menambah simpul bisa dipecah berdasarkan posisi simpul baru yang akan disisipkan, yaitu simpul baru selalau ditambahkan di belakang simpul terakhir, simpul baru selalu diletakkan sebagai simpul pertama, dan simpul baru menyisip di antara dua simpul yang sudah ada. Untuk menjelaskan operasi ini baiklah kita gunakan deklarasi **pointer** dan **simpul** seperti di bawah ini:

```
type Simpul = ^Data;
     Data    = record
                  Info      : char;
                  Berikut   : Simpul;
               end;
var  elemen                 : char;
     Awal, Akhir, Baru      : Simpul;
```

2. Menambah di Belakang

Operasi penambahan simpul pada suatu senarai berantai yang akan dijelaskan adalah penambahan simpul di akhir suatu senarai berantai. Dalam hal ini simpul-simpul baru yang ditambahkan selalu akan menjadi simpul terakhir. Ilustrasi penambahannya digambarkan seperti terlihat pada Gambar 5.3. **Pointer Awal** adalah

pointer yang menunjuk ke simpul pertama, **pointer Akhir** menunjuk ke simpul terakhir dan simpul yang ditunjuk oleh **pointer Baru** adalah simpul yang akan ditambahkan (Gambar 5.3.a.). Untuk menyambung simpul yang ditunjuk oleh **Akhir** dan **Baru**, **pointer** pada simpul yang ditunjuk oleh simpul **Akhir** dibuat sama dengan **Baru** (Gambar 5.3.b.), kemudian **pointer Akhir** dibuat sama dengan **pointer Baru** (5.3.c.).



Gambar 5.3. Ilustrasi penambahan simpul di belakang

Dengan memperhatikan Gambar 5.1, maka kita bisa menyusun prosedur untuk menambah simpul baru di belakang senarai berantai. Dalam hal ini perlu pula ditest apakah senarai berantai masih kosong atau tidak. Senarai berantai yang masih kosong ditandai dengan nilai **pointer Awal** yang nilainya sama dengan **nil**. Berdasarkan deklarasi simpul dan pointer di atas, maka prosedur untuk menambah simpul di belakang senarai berantai selengkapnya tersaji pada Program 1. di bawah ini.

Prosedur di bawah ini bisa dipanggil dengan pemanggil prosedur:

TAMBAH_BELAKANG (Awal, Akhir, Elemen);

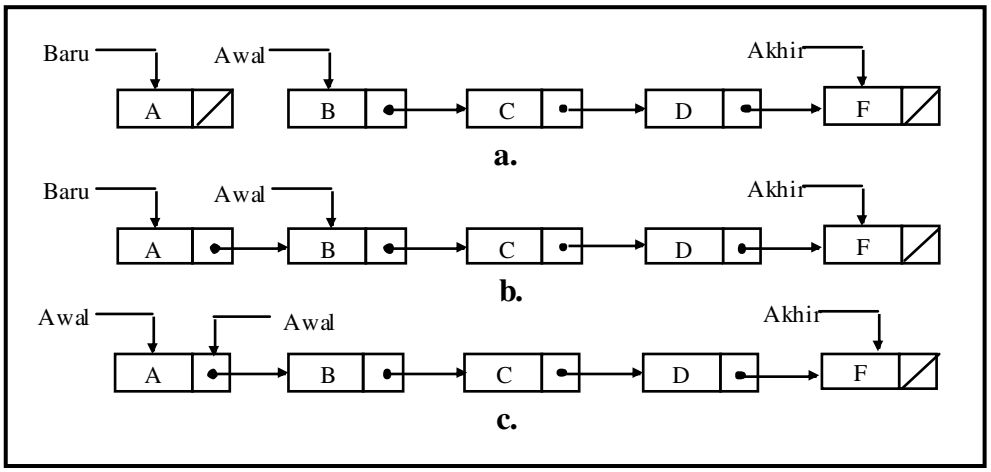
```
{*****************************************************
 * Prosedur untuk menambah simpul.                   *
 * Simpul baru diletakkan di akhir senarai berantai *
 *****************************************************}
 procedure TAMBAH_BELAKANG (var Awal; Akhir : Simpul;
                            Elemen : char);

 var Baru : Simpul;

 begin
    new (Baru); Baru^.Info := Elemen;
    if Awal = nil then         {*senarai masih kosong*}
       Awal := Baru
    else
       Akhir^.Berikut := Baru;         {*Gambar 1.b. *}
    Akhir := Baru;                     {*Gambar 1.c. *}
    Akhir^.Berikut :=nil
end;
```

Program 1. Prosedur menambah simpul di belakang senarai berantai

3. Menambah di Depan

Operasi penambahan simpul yang kedua adalah bahwa simpul baru akan selalu diletakkan di awal senarai berantai. Ilustrasi penambahannya disajikan pada Gambar 5.4. Secara garis besar operasi penambahannya bisa dijelaskan sebagai berikut. Pertama kali pointer pada simpul yang ditunjuk oleh **pointer baru** dibuat sama dengan awal (Gambar 5.4.b.). Kemudian, **Awal** dibuat sama dengan **Baru** (Gambar 5.4.c.). Dengan cara seperti ini simpul baru akan selalu diperlakukan sebagai simpul pertama dalam senarai berantai. Prosedur untuk menambah simpul di depan senarai berantai tersaji pada Program 2. Dalam hal perlu juga ditest apakah **Awal** masih **nil** atau tidak. Prosedur ini dipanggil menggunakan pemanggil prosedur.

TAMBAH_DEPAN (Awal, Akhir, Elemen);



Gambar 5.4. Ilustrasi penambahan simpul di awal senarai berantai

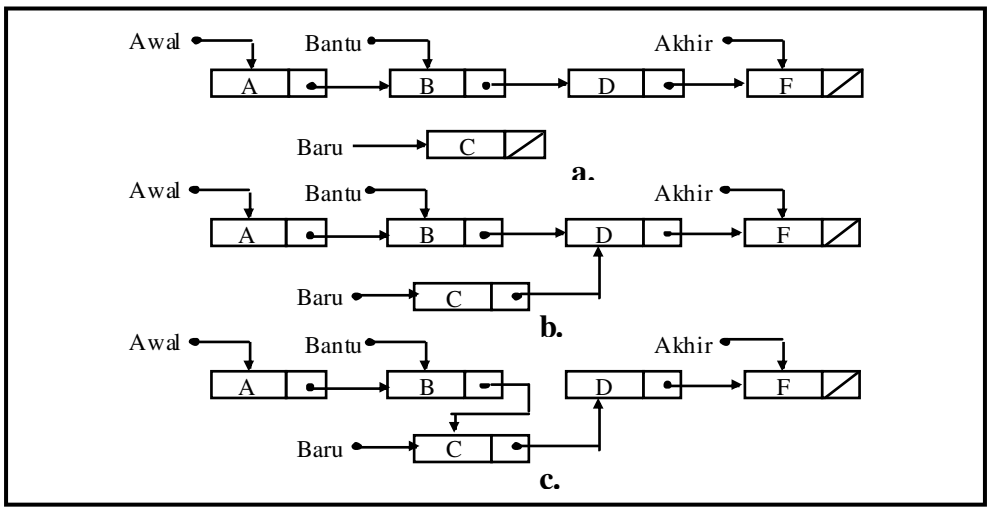
```
{*************************************************
 * Prosedur untuk menambah simpul.               *
 * Simpul baru diletakkan di awal senarai berantai  *
 *************************************************}
 procedure TAMBAH_DEPAN (var Awal; Akhir : Simpul;
                              Elemen : char);

 var Baru : Simpul;

 begin
    new (Baru); Baru^.Info := Elemen;
    if Awal = nil then       {*senarai masih kosong*}
       Akhir := Baru
 else
     Baru^.Berikut := Baru;          {*Gambar 2.b. *}
 Awal := Baru;                       {*Gambar 2.c. *}
```

Program 2. Prosedur menambah simpul di *awal* senarai berantai.

4. Menambah di Tengah

Untuk menambah simpul di tengah senarai berantai, kita perlu bantuan sebuah pointer lain, misalnya Bantu. Dalam hal ini simpul baru akan diletakkan setelah simpul yang ditunjuk oleh **pointer Bantu**. Ilustrasi penambahan seperti terlihat pada

Gambar 5.5 bisa dijelaskan sebagai berikut. Pertama kali ditentukan dimana simpul baru akan ditambahkan, yaitu dengan menempatkan **pointer Bantu** pada suatu tempat. Kemudian pointer pada simpul yang ditunjuk oleh **Baru** dibuat sama dengan pointer pada simpul yang ditunjuk oleh **Bantu** (Gambar 5.5.b.). Selanjutnya pointer pada simpul yang ditunjuk oleh simpul **Bantu** dibuat sama dengan **Baru** (Gambar 5.5.c.). Perhatikan bahwa urutan kedua proses ini tidak boleh dibalik. (Mengapa?). Prosedur selengkapnya adalah sebagai berikut. (Dalam hal ini senarai berantai terurutkan secara alphabetis, sehingga simpul baru yang akan ditambahkan harus dicari tempatnya yang sesuai).



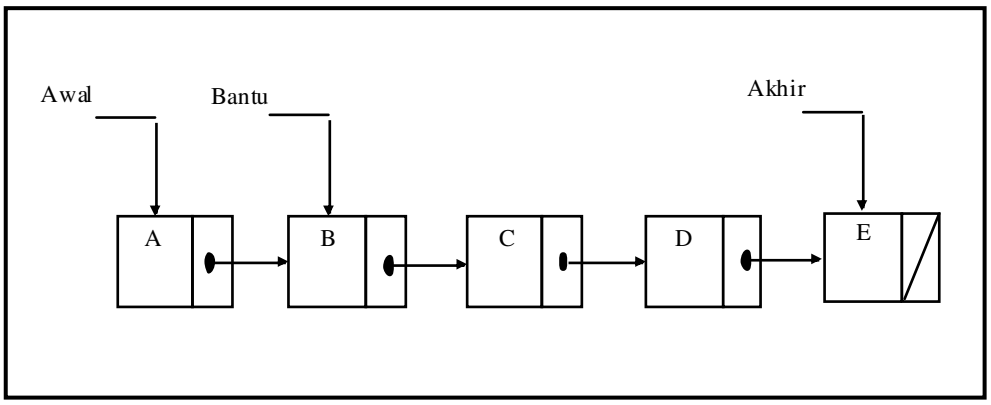
Gambar 5.5 Ilustrasi penambahan simpul di tengah senarai berantai

```
{* Prosedur untuk menambah simpul di tengah        *
* senarai berantai                                  * }
procedure TAMBAH_TENGAH (var Awal; Akhir : Simpul;
                         Elemen : char);
var Baru, Bantu : Simpul;
begin
    new (Baru); Baru^.Info := Elemen;
    if Awal = nil then       {*senarai masih kosong*}
       begin
         Awal := Baru;
         Akhir := Baru
       end
    else
       begin
          {* Mencari lokasi yang sesuai *}
          Bantu :=Awal;
          while Elemen > Baru^.Berikut^.Info do
               Bantu := Bantu^.Berikut;
          {* Menyisipkan = simpul baru *}
          Baru^.Berikut := Bantu^.Berikut {*Gambar 3.b. *}
          Bantu^.Berikut := Baru;         {*Gambar 3.c. *}
       end;
end;
```

Program 3. Prosedur menambah simpul di tengah senarai berantai

5. Menghapus Simpul

Operasi kedua yang akan dijelaskan adalah operasi menghapus simpul. Dalam menghapus simpul ada satu hal yang perlu diperhatikan, yaitu bahwa simpul yang bisa dihapus adalah simpul yang berada sesudah simpul yang ditunjuk oleh suatu **pointer** (dalam gambar adalah simpul yang berada di sebelah kanan simpul yang ditunjuk oleh suatu **pointer**), kecuali untuk simpul pertama. Dengan demikian, kita tidak bisa menghapus simpul yang ditunjuk oleh suatu **pointer** atau **simpul** sebelumnya. Untuk jelasnya perhatian ilustrasi berikut:



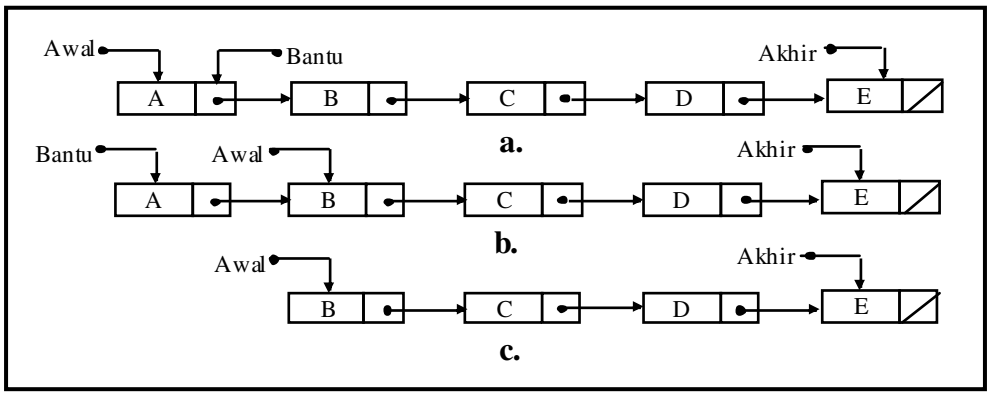
Gambar 5.6. Contoh senarai berantai

Jika senarai berantainya adalah seperti di atas, maka kita hanya bisa menghapus simpul yang berisi 'A' dan simpul yang berisi 'C' supaya senarai tetap bisa dipertahankan. Memang kita bisa menghapus simpul yang berisi 'B', tetapi senarai yang kita miliki akan menjadi terputus, karena akan lebih sulit untuk menyambung kembali simpu 'A' dengan simpul 'C'. Demikian juga jika kita menghapus simpul 'E', ada kemungkinan pointer pada simpul 'D' menjadi tidak menentu dan bisa menyebabkan terjadinya kesalahan.

Seperti halnya pada penambahan simpul, maka penghapusan simpul juga bisa kita pecah menjadi tiga macam penghapusan, menghapus simpul pertama, menghapus simpul yang ada di tengah dan menghapus simpul terakhir. Untuk menghapus simpul yang ada di tengah dan terakhir, kita perlu bantuan sebuah **pointer** yang menunjuk ke simpul sebelum (di sebelah kiri) simpul yang akan dihapus. Berikut akan dijelaskan masing-masing penghapusan ini, dan secara lengkap prosedurnya disajikan pada Program 4.

6. Menghapus Simpul Pertama

Untuk menghapus simpul pertama, maka **pointer Bantu** kita buat sama dengan **pointer Awal** (Gambar 5.7.a.). Kemudian **pointer Awal** kita pindah ke simpul yang ditunjuk oleh **pointer** pada simpul yang ditunjuk oleh **pointer Bantu** (Gambar 5.7.b.). selanjutnya, simpul yang ditunjuk oleh **pointer Bantu** kita **dispose** (Gambar 5.7.c.). Perhatikan gambar di bawah ini.

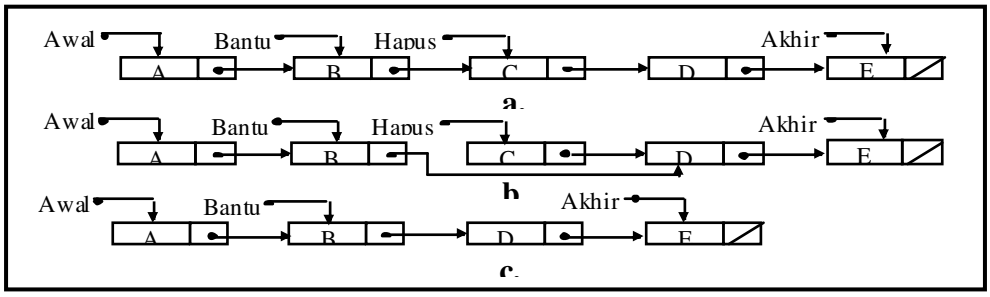


Gambar 5.7. Ilustrasi menghapus simpul pertama

7. Menghapus Simpul di Tengah atau Terakhir

Untuk menghapus simpul yang ada di tengah senarai berantai, pertama kali kita letakkan **pointer Bantu** pada simpul di sebelah kiri simpul yang akan dihapus. Simpul yang akan dihapus kita tunjuk dengan pointer lain, misalnya **Hapus** (Gambar 5.8.a.). Kemudian **pointer** pada simpul yang ditunjuk oleh **Bantu** kita tunjukkan pada simpul yang ditunjuk oleh pointer pada simpul yang akan dihapus (Gambar 5.8.b.). Selanjutnya simpul yang ditunjuk oleh pointer **Hapus** kita **dispose** (Gambar 5.8.c.). Prosedur selengkapnya tersaji dalam Program 4. Perubah **Elemen** berisi data yang akan dihapus. Prosedur ini dipanggil dengan statemen:

HAPUS_SIMPUL (Awal, Akhir, Elemen);



Gambar 5. 8. Ilustrasi menghapus simpul yang ada di tengah senarai berantai

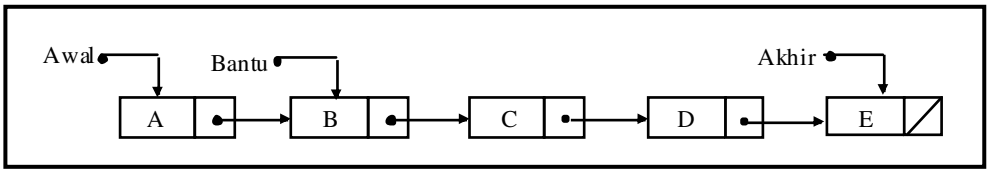
```
{*************************************************
 * Prosedur menghapus simpul                     *
 *************************************************}
 procedure HAPUS_SIMPUL (var Awal, Akhir : Simpul;
                             Elemen : char);
  var Bantu, Hapus : Simpul;
 begin
   if Awal = nil then        {* senarai masih kosong*}
       writeln ('Senarai masih kosong')

   else if Awal^.Info = Elemen then
        {* simpul pertama dihapus *}
        begin
          Hapus := Awal;
          Awal := Hapus^.Berikut;
          dispose (Hapus)
        end

   else
        {* menghapus simpul tengah atau terakhir *}
        begin
          Bantu :=Awal;
          {* Mencari simpul yang akan dihapus *}
          while (Elemen <> Bantu^.Info) and
                (Bantu^.Berikut <> nil) do
          Bantu := Bantu^.Berikut;
          Hapus := Bantu^.Berikut;
          If Hapus <> nil then
          { * Simpul yang akan dihapus ketemu * }

          begin
              if Hapus <> Akhir then
              { *Simpul tengah di hapus * }
              Bantu^.Berikut := Hapus^.Berikut
              Else
                 {*Simpul terakhir dihapus*}
                 begin
                     Akhir := Bantu;
                     Akhir^.Berikut := nil
                 End;
              Dispose (Hapus)
          End
          Else
            {*Simpul yang akan dihapus tidak ketemu*}
      writeln('Simpul yang akan dihapus tidak ada')


       end
end;
```

Program 4. Prosedur menambah simpul di tengah senarai berantai.

8. Membaca Isi Senarai Berantai

Jika kita mempunyai senarai berantai seperti yang telah kita pelajari, maka dengan cara biasa kita hanya bisa membaca isi simpul dimulai dari simpul paling kiri sampai simpul paling kanan (membaca maju). Untuk bisa membaca dari kanan ke kiri

(membaca mundur) bisa dilaksanakan dengan dua cara. Hal ini akan dijelaskan pada sub-bab berikut:



Gambar 5.9. Ilustrasi membaca senarai berantai dari kiri ke kanan (membaca maju).

9. Membaca Maju

Pembaca senarai berantai secara maju bisa dijelaskan sebagai berikut (perhatikan Gambar 5.9.). Pertama kali kita atur supaya **pointer Bantu** menunjuk ke simpul yang ditunjuk oleh **pointer Awal**. Setelah isi simpul tersebut dibaca, maka **pointer Bantu** kita gerakkan ke kanan untuk membaca simpul berikutnya. Proses ini kita ulang sampai **pointer Bantu** sama dengan **pointer A**khir. Prosedur untuk membaca senarai berantai dari kiri ke kanan (membaca maju) tersaji di bawah ini, dan dipanggil dengan statemen:

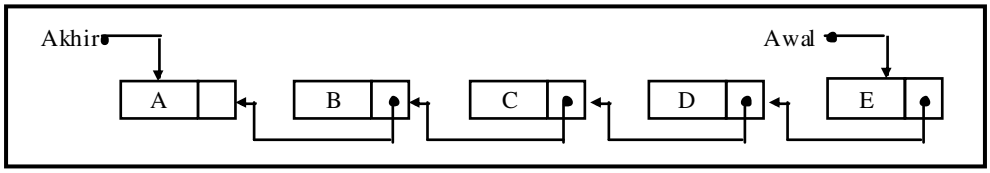
BACA_AMJU (Awal, Akhir);

```
{*************************************************
 * Prosedur untuk membaca senarai dari           *
 * kiri ke kanan (maju)                          *
 *************************************************}
 procedure BACA_MAJU (Awal; Akhir : Simpul);
 var
    Bantu : Simpul;
 Begin
      Bantu := Awal;
      Repeat
            Write(Bantu^.Info:2);
            Bantu := Bantu^.Berikut
      Until
            Bantu = nil;
      Writeln
 End;
```

Program 5. Prosedur membaca senarai berantai dari kiri ke kanan (membaca maju).

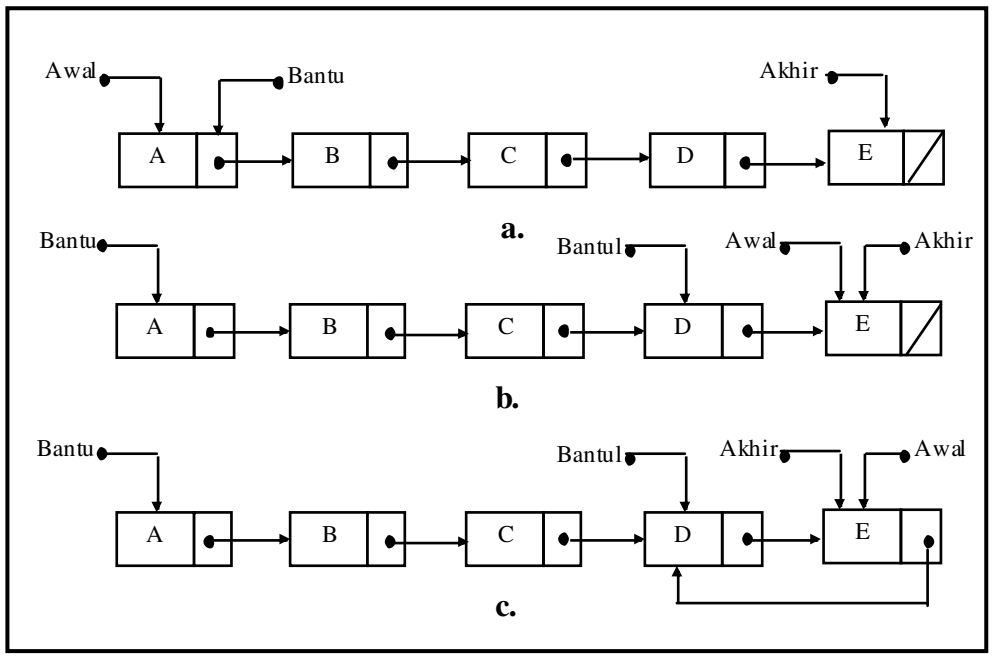
10. Membaca Mundur

Untuk membaca dari kanan ke kiri (membaca mundur), untuk memperoleh untai "EDCBA" dari Gambar 5.9, ada dua cara yang bisa kita lakukan. Cara pertama adalah dengan cara biasa seperti di atas dan cara kedua adalah dengan mengunakan proses rekursi. Marilah kita melihat pada kedua cara ini satu persatu. Untuk cara pertama, maka senarai berantai Gambar 5.9. harus diubah menjadi seperti gambar di bawah ini:



Gambar 5.10. Ilustrasi membaca senarai berantai dari kanan ke kiri (membaca mundur).

Langkah-langkah untuk memperoleh keadaan dari Gambar 5.9 menjadi Gambar 5.10 bisa digambarkan seperti terlihat pada Gambar 5.11. Ilustrasi tersebut bisa dijelaskan sebagai berikut. Pertama kali kita atur **pointer Bantu** sama dengan **pointer Awal** (Gambar 5.11.a.). Berikutnya, **pointer awal** kita arahkan ke simpul terakhir; dan kita pakai pointer lain, **pointer Bantul** untuk bergerak dari simpul pertama sampai simpul sebelum simpul terakhir (Gambar 5.11.b.). Langkah berikutnya adalah mengarahkan medan pointer dari simpul **Akhir** ke simpul **Bantul**. Langkah ini diulang sampai **pointer Akhir** berimpit dengan **pointer Bantu.**



Gambar 5.11. Ilustrasi menbalik pointer.

Prosedur untuk membalik pointer tersaji pada Program 6. berikut ini. Dengan menggunakan prosedur tersebut maka senarai berantai yang semula seperti pada Gambar 5.9. akan menjadi seperti Gambar 5.10. Setelah pointer dibalik seperti yang kita inginkan (Gambar 5.10), maka kita bisa mencetaknya menggunakan cara yang sudah dijelaskan sebelumnya. Cara kedua adalah dengan proses rekursi. Caranya memang sama seperti dijelaskan pada Gambar 5.9. di atas, tetapi pencetakan isi simpul ditunda dahulu sampai pointer Bantu sama dengan pointer Akhir. Dengan cara seperti ini maka akan kita peroleh untai seperti yang kita harapkan. Prosedur yang tersaji pada Program 7. adalah prosedur untuk membaca senarai berantai dari kanan ke kiri (membaca mundur) secara rekursif. Perhatikan isi parameter pada nama prosedur. Prosedur tersebut dipanggil dengan statamen:

MUNDUR (Awal):

```
{*************************************************
 * Prosedur untuk membalik pointer               *
 *************************************************}
 procedure BALIK_POINTER (var Awal; Akhir : Simpul);
 var
      Bantu, Bantul : Simpul;
 Begin
      Bantu := Awal;
      Awal := Akhir;
      {*Proses membanlik pointer*}
      repeat
            Bantul := Bantu;
            {*Mencari simpul sebelum simpul yang ditunjuk
              oleh pointer Akhir *}
            while Bantul^.Berikut <> Akhir do
                Bantul := Bantul^.Berikut;
            Akhir^.Berikut := Bantul := Bantul;
            Akhir := Bantul;
      until
            Akhir = Bantu;
            Akhir^.Berikut := nil;
End;
```

Program 6. Prosedur untuk membalik pointer.

11. Mencari Data

Secara garis besar, proses untuk mencari data yang terdapat pada suatu simpul hampir sama dengan proses membaca isi simpul, tetapi perlu ditambah dengan test untuk menentukan apakah data yang dicari ada pada senarai berantai. Dengan mengacu pada Gambar 9,. maka proses pencarian data bisa dijelaskan sebagai berikut. Misalkan data yang dicari disimpan dalam perubah Elemen. Pertama kali, **pointer Bantu** dibuat sama dengan **pointer Awal**. Kemudian isi simpul yang ditunjuk oleh **pointer Bantu** dibandingkan dengan **Elemen**. Jika belum sama, maka pointer bantu dipindah ke simpul di sebelah kanannya, dan proses pembandingan diulang lagi. Proses ini diteruskan sampai bisa ditentukan apakah **Elemen** ada dalam senarai yang dimaksud.

```
{*************************************************
 * Prosedur untuk membaca senarai dari kanan ke  *
 * kiri menggunakan cara pembacaan secara rekursif *
 *************************************************}
 procedure MUNDUR (Bantu : Simpul);
 begin
       if Bantu <> nil then
       begin
            MUNDUR (Bantu^.Berikut);
            Write (Bantu^.Info:2)
       End;
 End;
```

Program 7. Membaca senarai berantai dari kanan ke kiri (membaca mundur) secara rekursif.

```
      {*************************************************
           * Fungsi untuk Mencari data pada senarai tak    *
           * terurutkan jika fungsi bernilai "true" berarti*
       * data ketemu.  *

      *************************************************}
           Function      ADA_DATA      (Elemen      :      char;
Awal:Simpul):Boolean;
      Var
         Ketemu : Boolean;
      Begin
           Ketemu := false;
           Repeat
                  If Awal^.Info = Elemen then
                     Ketemu := true
                  Else
                     Awal := Awal^.Berikut;
           Until
                  Ketemu or (Awal = nil);
           ADA_DATA := ketemu
      End;
```

Program 8. Fungsi untuk mencari data pada senarai berantai tak terurutkan

Program 8. menyajikan contoh fungsi untuk mencari data pada sebuah senarai berantai. Dalam hal ini senarai berantai tidak dalam keadaan urut, sehingga dalam keadaan yang paling buruk, pencarian harus dilakukan sampai akhir senarai berantai dicapai. Fungsi ini dibuat dengan tipe **boolean.** Fungsi akan bernilai **true** jika data yang dicari ada, dan bernilai **false** jika data yang dicari tidak ada. Fungsi yang disajikan berikut ini hanya mencek apakah data yang dicari ketemu atau tidak. Jika senarai berantai yang akan kita cari datanya sudah dalam keadaan terurutkan, misalnya secara urut naik atau alphabetis, maka ada satu keuntungan yang kita peroleh. Keuntungan tersebut adalah bahwa jika nilai **Elemen** telah lebih besar dari nilai informasi pada simpul yang kita tinjau, maka proses pencarian bisa dihentikan, karena jelas bahwa informasi yang ingin kita cara memang tidak ada dalam senarai berantai tersebut. Dengan demikian fungsi di atas perlu kita sesuaikan, yaitu menjadi:

```
     {*************************************************
      * Fungsi untuk mencari data pada senarai terurutkan  *
      * jika fungsi bernilai 'true' berarti data ketemu.   *
      *************************************************}
      Function ADA_DATA1 (Awal; Simpul; Elemen : char): Boolean;
      Var
         Ketemu : Boolean;
      Begin
           Ketemu := false;
           Repeat
                  If Awal^.Info = Elemen then
                     {* Data ditemukan*}
                     ketemu := true;
                  else
                     if Awal^.Info < Elemen then
                        {* belum ditemukan*}
                        Awal := Awal^.Berikut
                     Else
                        {*tidak ditemukan*}
                        Bantu := nil
           Until
                  Ketemu or (Awal = nil);
           ADA_DATA1 := ketemu
      End;
```

Program 9. Fungsi untuk mencari data pada senarai berantai terurutkan.

Karena sifatnya adalah fungsi, maka harus dipanggil dalam suatu statemen pemberian atau testing, misalnya.

Data_Ketemu = ADA_DATA (Elemen, Awal);

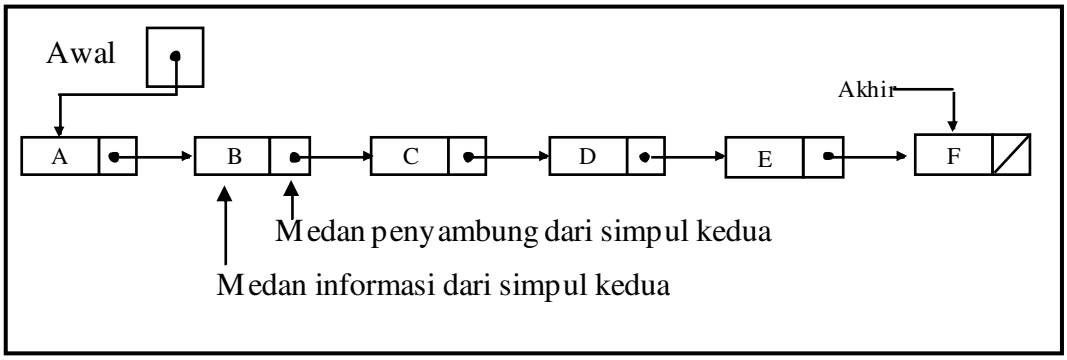
atau

if ADA_DATA (Elemen, Awal) then

dengan Data_Ketemu adalah data bertipe **boolean** dan **Elemen** adalah data yang akan dicari.

5.3 Contoh Penyajian Senarai Berantai

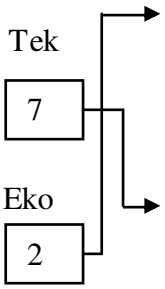
Mari kita lihat bagaimana sesungguhnya suatu senarai berantai disimpan dalam memori. Di atas telah disinggung bahwa meskipun simpul-simpul dalam senarai berantai secara skematis bisa digambarkan secara berurutan, tetapi dalam memori simpul-simpul yang berurutan tidak harus disimpan secara berurutan pula, karena sesungguhnya pointer adalah alamat lokasi dalam memori. Misalkan senarai berantai dalam Gambar 5.12 berikut, dinamakan DAFTAR.



Gambar 5. 12. Contoh senarai berantai.

Senarai berantai DAFTAR ini dalam memori akan disajikan dengan cara sebagai berikut. Pertama-tama DAFTAR memerlukan dua larik linear (vektor), kita namakan vektor **INFO** berisi komponen yang tersimpan dalam medan informasi dan vektor SAMBUNGAN berisi medan **pointer** (alamat) ke simpul berikutnya. Disamping itu, DAFTAR juga memerlukan suatu perubah, misalnya **AWAL** yang berisi alamat simpul pertama dari senarai berantai, dan **pointer** yang bernama **KOSONG**, yang menunjukkan akhir sebuah senarai berantai. Karena subskrib untuk vektor **INFO** dan SAMBUNGAN selalu positif dan lebih besar dari 0, maka untuk **pointer** bisa kita tulis sebagai KOSONG = 0

Contoh lain misalnya sebagai berikut. Data mahasiswa dari dua Fakultas, misalnya Fakultas Teknik dan Ekonomi, akan disimpan pada senarai berantai yang strukturnya persis sama, yaitu terdiri dari data: Nama mahasiswa, Angkatan dan jenis kelamin. Gambar 14, di bawah ini menunjukkan salah satu cara penyajian senarai berantai dari mahasiswa pada kedua Fakultas di atas; yaitu bahwa Nama, Nomor, Angkatan dan Jenis kelamin untuk semua mahasiswa dari dua senarai yang berbeda disimpan pada sebuah vektor yang sama, yaitu vektor, NAMA, NOMOR, ANGK, dan JENIS KELAMIN. Untuk membedakan kedua senarai tersebut diperlukan dua **pointer Awal** yang berbeda, misalnya Tek dan Eko.



	NAMA	NOMOR	ANGK	JENIS KELAMIN	SAMBUNGAN
1	TONI SUBAGIO	12345	84/85	L	9
2	BUDIMAN	6745	86/87	L	6
3					
4	PURWANTO	7001	85/86	L	0
5	BAMBANG	12123	83/84	L	11
6	ESTIYANTI	6501	84/85	P	13
7	AMIR	12211	83/84	L	5
8					
9	WIDODO	12111	83/84	L	0
10					
11	DANANG	12432	84/85	L	1
12					
13	JATI	7201	86/87	L	4
14					

Gambar 5.13. Penyajian dua buah senarai berantai.

Dari Gambar tersebut bisa dilihat bahwa mahasiswa Fakultas Teknik antara lain adalah :
AMIR, BAMBANG, DANANG, TONI, SUBAGIO, WIDODO.
Dan mahasiswa Fakultas Ekonomi antara lain adalah:
BUDIMAN, ESTYANTI, JATI, PURWANTO.

5.4 Contoh program senarai berantai :

```
Program Senarai_Berantai;
uses crt;
type
        LinkList=^List;
                List=record
                Nama:string;
                Jabatan:string;
                Telp:integer;
                next:LinkList;
        end;
var
        awal,baru,r:LinkList;
        j:integer;
begin
        clrscr;
        new(baru);
        write('Masukkan Nama   : ');
        readln(baru^.Nama);
        write('Masukkan Jabatan: ');
        readln(baru^.Jabatan);
        write('Masukkan Telp   : ');
```

```
        readln(baru^.Telp);
        awal:=baru;
        r:=baru;
        for j:=2 to 3 do
        begin
                new(baru);
                write('Masukkan Nama   : ');
                readln(baru^.Nama);
                write('Masukkan Jabatan: ');
                readln(baru^.Jabatan);
                write('Masukkan Telp   : ');
                readln(baru^.Telp);
                r^.next:=baru;
                r:=r^.next;
        end;
        baru^.next:=nil;
        r:=awal;
        for j:=1 to 3 do
        begin
                writeln('Nama    : ',r^.Nama);
                writeln('Jabatan : ',r^.Jabatan);
                writeln('Telp    : ',r^.Telp);
                r:=r^.next         ;
        end;
        readln;
end.
```

==========

output program tersebut sebagai berikut:

```
Masukkan Nama     : budi
Masukkan Jabatan  : dosen
Masukkan Telp     : 5810
Masukkan Nama     : rudi
Masukkan Jabatan  : guru besar
Masukkan Telp     : 5810
Masukkan Nama     : joko
Masukkan Jabatan  : asisten dosen
Masukkan Telp     : 3980
```

==========

```
Nama    : budi
Jabatan : dosen
Telp    : 5810
Nama    : rudi
Jabatan : guru besar
Telp    : 5811
Nama    : joko
Jabatan : asisten dosen
Telp    : 3980
```

BAB 6
POINTER

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi pointer, deklarasi pointer dan alokasi tempat, operasi pada pointer dan disertai dengan contoh program pointer

Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi pointer, deklarasi pointer dan alokasi tempat, operasi pada pointer dan dapat membuat program pointer.

PENYAJIAN

Pemakaian larik tidak selalu tepat untuk program-program terapan yang kebutuhan pengingatnya/memorinya selalu bertambah selama eksekusi program tersebut. Untuk itu diperlukan satu tipe data yang bisa digunakan untuk mengalokasikan dan mendealokasikan pengingat secara dinamis, yaitu sesuai dengan kebutuhan pada saat suatu program dieksekusi. Pada kuliah ini akan menjelaskan tipe data yang disebut dengan tipe pointer yang bisa dialokasikan dan didealokasikan sesuai kebutuhan.

6. 1 Pengertian Pointer

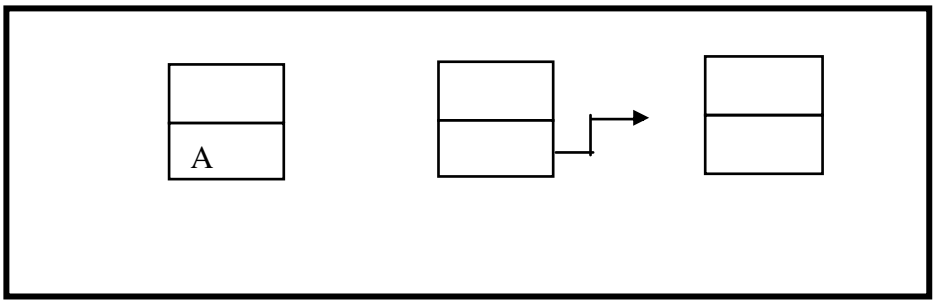
Nama perubah, yang kita gunakan untuk mewakili suatu nilai data, sebenarnya merupakan atau menunjukkan suatu lokasi tertentu dalam pengingat komputer dimana data yang diwakili oleh nama perubah tersebut disimpan. Pada saat sebuah program dikompilasi, kompiler akan melihat pada bagian deklarasi perubah (**var**) untuk mengetahui nama-nama perubah apa saja yang akan digunakan, sekaligus

mengalokasikan atau menyediakan tempat dalam pengingat untuk menyimpan nilai data tersebut. Dari sini kita bisa melihat bahwa sebelum program dieksekusi (harap dibedakan antara kompilasi dan eksekusi program), lokasi-lokasi data dalam pengingat sudah ditentukan dan tidak bisa diubah selama program tersebut dieksekusi. Perubah-perubah yang demikian ini dinamakan dengan perubah statis (*static variable*). Dari pengertian di atas bisa kita perhatikan bahwa sesudah suatu lokasi pengingat ditentukan untuk suatu nama perubah, maka dalam program tersebut perubah yang dimaksud akan tetap menempati lokasi yang telah ditentukan dan tidak mungkin diubah. Dengan melihat pada sifat-sifat perubah statis, bisa kita katakan bahwa banyaknya data yang bisa diolah adalah terbatas. Sebagai contoh, misalnya kita mempunyai perubah yang kita deklarasikan sebagai:

var Tabel : array[1..100, 1..50] of integer;

Perubah Tabel di atas hanya mampu untuk menyimpan data sebanyak 100 x 50 = 5000 buah data. Jika kita tetap nekat untuk menambah data pada perubah tersebut, eksekusi program akan terhenti karena deklarasi lariknya kurang. Memang kita bisa mengubah deklarasi di atas untuk memperbesar ukurannya. Tetapi jika setiap kali kita harus mengubah deklarasi di atas, sementara banyaknya data tidak bisa kita tentukan lebih dahulu, hal ini tentu merupakan pekerjaan yang **membosankan**. Sekarang bagaimana jika kita ingin mengolah data yang banyaknya tidak bisa kita pastikan sebelumnya, sehingga kita tidak yakin dengan deklarasi larik kita?

Untuk menjawab pertanyaan di atas, Pascal menyediakan satu fasilitas yang memungkinkan kita menggunakan suatu perubah yang disebut dengan perubah dinamis (*dynamic variable*). Perubah dinamis adalah suatu perubah yang akan dialokasikan hanya pada saat diperlukan, yaitu setelah program dieksekusi. Dengan kata lain, pada saat program dikompilasi, lokasi untuk perubah tersebut belum ditentukan. Kompiler hanya akan mencatat bahwa suatu perubah akan diperlakukan sebagai perubah dinamis. Hal ini membawa keuntungan pula, bahwa perubah-perubah dinamis tersebut bisa dihapus pada saat program dieksekusi, sehingga ukuran pengingat akan selalu berubah. Hal inilah yang menyebabkan perubah dinamakan sebagai perubah dinamis. Pada perubah statis, isi pengingat pada lokasi tertentu (nilai perubah) adalah data sesungguhnya yang akan kita olah. Pada perubah dinamis, nilai perubah adalah alamat lokasi lain yang menyimpan data yang sesungguhnya. Dengan demikian data yang sesungguhnya tidak bisa dimasup secara langsung. Untuk memperjelas pengertian di atas, bayangkan ilustrasi berikut ini. Seseorang bernama A ingin menelpon temannya yang bernama B. Dalam hal ini nomor telpon B sudah diketahui A, sehingga ia bisa langsung menelponnya. Pada hari lain A ingin menelpon C, tetapi A tidak tahu nomor telpon C, sehingga tentu saja ia tidak bisa menelpon langsung. Untuk bisa menelpon C, maka A harus membuka catatan nomor telponnya atau menanyakannya pada orang lain yang mengetahui nomor telpon C. Setelah informasi nomor telpon C diperoleh A, A bisa menelpon C. Dalam hal cara pemasupan data, kasus pertama pada ilustrasi di atas mirip dengan pengertian perubah statis dan kasus kedua mirip dengan pengertian perubah dinamis. Cara pemasupan data bisa diilustrasikan seperti Gambar (1) di bawah ini:



Gambar 6.1. Ilustrasi perubah statis dan dinamis

Gambar 6.1 di atas bisa dijelaskan sebagai berikut. Pada Gambar 6.1a, perubah A adalah perubah statis. Dalam hal ini 1000 adalah nilai data yang sesungguhnya dan disimpan pada perubah (lokasi) A. Pada Gambar 6.1b, perubah A adalah perubah dinamis. Nilai perubah ini, misalnya adalah 10. Nilai ini bukan nilai data yang sesungguhnya, tetapi lokasi dimana data yang sesungguhnya berada. Jadi dalam hal ini nilai data yang sesungguhnya tersimpan pada lokasi 10. Dari ilustrasi di atas bisa dilihat bahwa nilai perubah dinamis akan digunakan untuk menunjuk ke lokasi lain yang berisi data sesungguhnya yang akan diproses. Karena alasan inilah perubah dinamis lebih dikenal dengan sebutan **pointer** yang artinya kira-kira menunjuk ke sesuatu. Dalam perubah dinamis, nilai data yang ditunjuk oleh suatu **pointer** biasanya disebut sebagai **simpul/*node***.

6.2 Deklarasi Pointer dan Alokasi Tempat

Seperti halnya dengan tipe data yang lain, tipe **pointer** biasanya dideklarasikan pada bagian deklarasi **type**.

Bentuk umum deklarasi **pointer** adalah:

Type pengenal **= ^** ***simpul;***
Simpul = tipe;

dengan *pengenal* : nama pengenal yang menyatakan data bertipe pointer
simpul : nama simpul
tipe : tipe data dari simpul

Tanda ^ di depan nama *simpul* harus ditulis seperti apa adanya dan menunjukkan bahwa *pengenal* adalah suatu tipe data **pointer**. Tipe data ***simpul*** yang dinyatakan dalam ***tipe*** bisa berupa sembarang tipe data, misalnya **char, integer, atau real**. Sebagai contoh:

type Bulat = ^integer;

Dalam contoh di atas **Bulat** menunjukkan tipe data baru, yaitu bertipe **pointer**. Dalam hal ini **pointer** tersebut akan menunjuk ke suatu data yang bertipe **integer**. Dengan deklarasi di atas, maka kita bisa mempunyai deklarasi perubah, misalnya:

var X, K : Bulat;

yang menunjukkan bahwa X, dan K adalah perubah bertipe **pointer** yang hanya bisa memasup data yang bertipe **integer**. Dalam kebanyakan program-program terapan, biasanya terdapat sekumpulan data yang dikumpulkan dalam sebuah rekaman (**record**), sehingga anda akan banyak menjumpai tipe data **pointer** yang elemennya (data yang ditunjuk) adalah sebuah rekaman. Perhatikan contoh berikut:

```
type  Str30  = string[30];
      Point = ^Data;
      Data   = record
                  Nama_Peg: Str30;
                  Alamat     : Str30;
                  Pekerjaan : Str30;
               end;
```

Dengan deklarasi tipe data seperti di atas, kita bisa mendeklarasikan perubah, misalnya:

```
Var
      P1, p2          : point;
      a, b, c         : Str30;
```

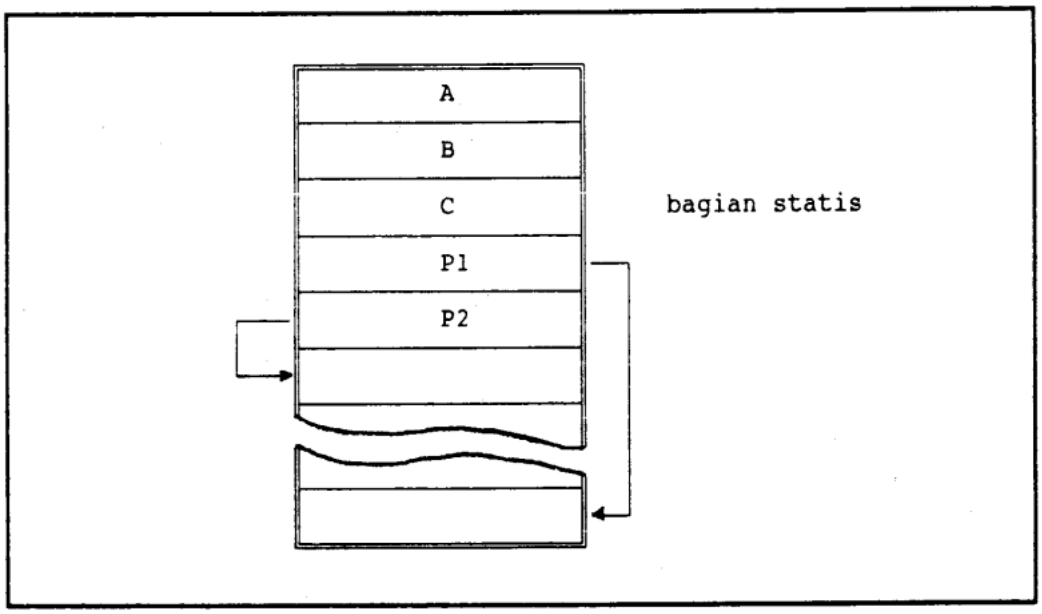
Pada contoh di atas, p1 dan p2 masing-masing bertipe **pointer**; a, b, dan c perubah statis yang bertipe **string**. Pada saat program dikompilasi, perubah p1 dan p2 akan menempati lokasi tertentu dalam memori. Kedua perubah ini masing-masing belum menunjuk ke suatu simpul. **Pointer** yang belum menunjuk ke suatu simpul nilainya dinyatakan sebagai **nil**. Untuk mengalokasikan simpul dalam memori, statemen yang digunakan adalah statemen **new**, yang mempunyai bentuk umum:

```
New (perubah)
```

Dengan perupah yang bertipe **pointer**. Sebagai contoh, dengan deklarasi perubah seperti di atas dan statemen:

```
New (p1);
New (p2);
```

Maka sekarang kita mempunyai dua buah simpul yang ditunjuk oleh p1 dan p2. Jika kita lihat dalam memori, maka alokasi perubah di atas adalah:

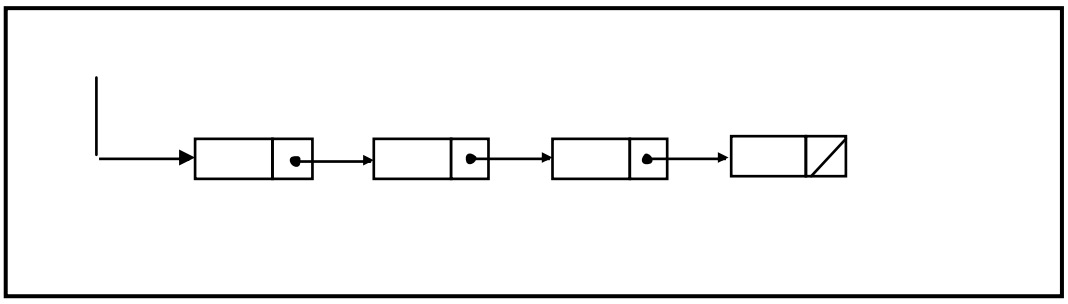


Gambar 6.2 contoh alokasi memori.

Dalam contoh di atas, banyak memori yang diperlukan pada dasarnya juga tetap. Lebih lanjut, jika statemen:

New (p1)

Diulang beberapa kali, maka hanya simpul terakhir yang bisa dimasup. Hal ini terjadi karena setiap kali kita memberikan statemen **new (p1)**, maka nilai p1 yang lama akan terhapus. Dengan terhapusnya nilai p1 secara otomatis simpul yang ditunjukkan oleh p1 tidak ditunjukkan, maka **simpul** tersebut tidak bisa dimasup lagi. Dengan melihat pada contoh di atas, bisa kita perhatikan, bahwa deklarasi di atas sifat kedinamisannya masih tersamar. Alasannay adalah kita menghendaki sejumlah **simpul** aktif dalam memori, maka kita perlu menyediakan sejumlah **pointer** yang sesuai. Dengan demikian seolah-olah tidak adal perbedaan yang nyata antara perubah **statis** dengan **pointer**. Jika kita benar-benar ingin mempunyai suatu perubah yang benar-benar bersifat dinamis, maka kita harus mampu untuk memasup sejumlah lokasi tetentu dengan hanya menggunakan sebuah **pointer** awal, misalnya seperti pada Gambar 6.3 di bawah ini.



Gambar 6.3 simpul-simpul yang membentuk senarai berantai.

Pada Gambar di atas, p1 adalah perubah yang bertipe **pointer** dengan simpulnya bertipe rekaman. Salah satu medan dalam simpul tersebut juga bertipe **pointer** dengan tipe data yang sama dengan p1, sehingga dengan memanfaatkan deklarasi tipe **pointer** ini, bentuk senarai berantai seperti Gambar 6.3 di atas bisa diperoleh dengan mudah. Untuk membentuk keadaan di atas, maka deklarasi tipe **pointernya** kita ubah menjadi:

```
Type
      Perubah  = ^ simpul;
      Simpul   = record
                    Info      : tipe;
                    Berikut   : perubah
                 End;
```

Dengan perubah : nama perubah yang bertipe pointer.
Simpul : nama simpul.
Info : nama medan dari data yang bisa bertipe sembarang.
Tipe : tipe data dari masing-masing medan.
Berikut : nama medan yang bertipe data pointer.

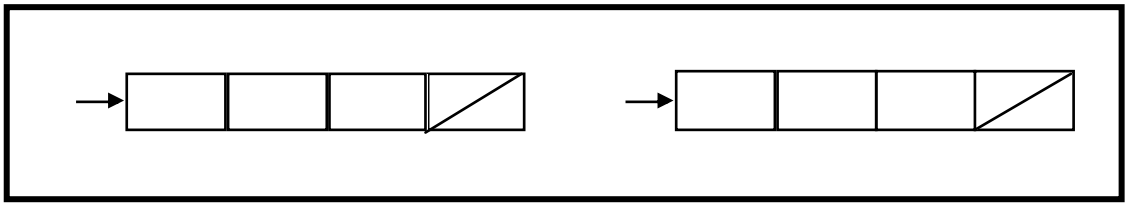
Sebagai contoh, jika kita mempunyai deklarasi data:

```
Type
      Str30  = string[30];
      Point  = ^data;
      Data   = record
                          Nama_peg   : str30;
                          Alamat     : str30;
                          Pekerjaan  : str30;
                          Berikut    : point;
               End;
Var
      P1, p2 : point;
```

Maka setelah statemen:

```
New (p1);
New (p2);
```

Dieksekusi, kita mempunyai dua buah simpul yang apabila kita gambar adalah seperti tersaji pada Gambar 6.4.



Gambar 6.4. simpul yang berisi medan bertipe pointer.

Perhatikan bahwa, karena medan nama, alamat, dan pekerjaan belum mempunayi nilai, maka nilainya ditunjukkan dengan tanda ?. Sedangkan pada medan yang bertipe **pointer** (medan berikut), karena ia tidak menunjuk ke simpul lain, maka nilainya adalah **nil**, yang disimbolkan seperti gambar di atas. Dengan keadaan seperti di atas, maka kita akan lebih mudah untuk menggandeng simpul-simpul yang ditunjuk oleh p1 dan p2; sehingga dengan menggunakan **pointer** p1 atau p2 saja kita bisa memasup dua buah simpul ini.

6.3 Operasi pada pointer

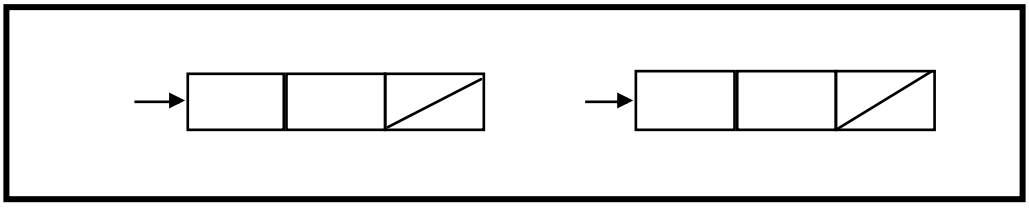
Secara umun ada dua operasi dasar bisa kita lakukan menggunakan data yang bertipe **pointer**. Operasi yang pertama adalah **mengkopi pointer**, sehingga sebuah simpul akan ditunjuk oleh lebih dari sebuah **pointer**. Operasi kedua adalah **mengkopi isi simpul**, sehingga dua atau lebih simpul yang ditunjuk oleh **pointer** yang berbeda mempunyai isi yang sama. Syarat yang harus dipenuhi untuk kedua operasi ini adalah bahwa **pointer-pointer** yang akan dioperasikan harus mempunyai deklarasi yang sama. Untuk memahami kedua operasi dasar di atas, perhatikan contoh berikut. Pertama kali kita deklarasikan tipe **pointer**, yaitu:

```
Type
    Simpul = ^data
    Data   = record
                Nama      : string;
                Alamat    : string;
                Berikut   : simpul
             End;
  Var
    t1, t2 : simpul;
```

Pada deklarasi di atas, **pointer** t1 dan t2 mempunyai deklarasi simpul yang sama, sehingga memenuhi syarat untuk kedua operasi di atas. Sekarang, jika kita memberikan statemen:

```
New (t1);
New (t2);
```

Kita mempunyai dua simpul, yaitu:

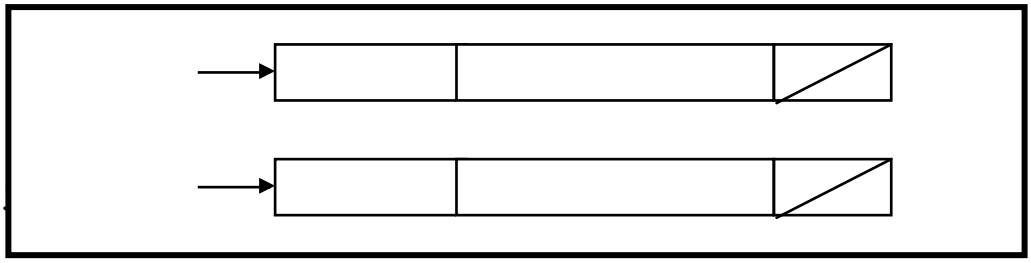


Gambar 6.5 contoh dua simpul.

Dengan menggunakan statemen:

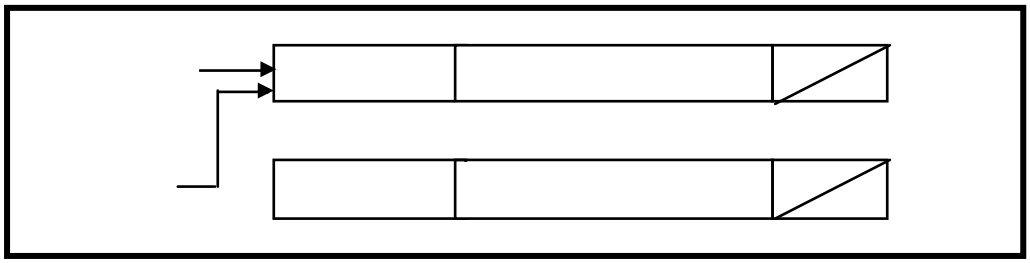
```
t1^.nama := 'OKKY';
t1^.alamat := 'ACEH UTARA';
```

Maka keadaan dua simpul di atas berubah menjadi:



t2 := t1;

Maka gambar (6) di atas berubah menjadi:

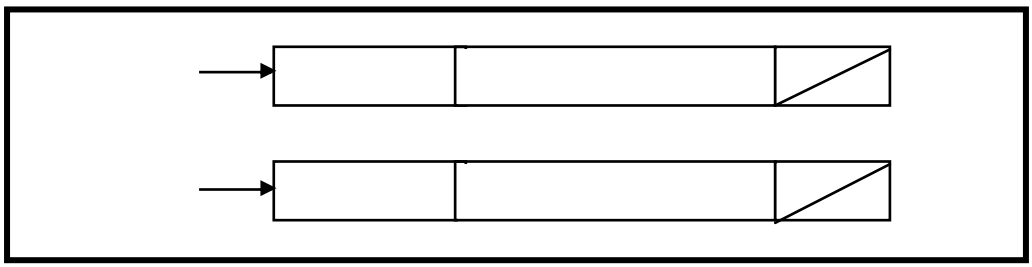


Gambar 6.7 keadaa baru dari gambar 6.6

Dari gambar di atas bisa kita perhatikan, bahwa sekerang **pointer** t2 juga menujuk ke simpul yang ditunjuk oleh **pointer** t1; simpul yang semula ditunjuk oleh t2, menjadi terlepas. Dalam keadaan seperti ini, karena tidak ditunjuk oleh suatu **pointer** lain, maka simpul tersebut tidak bisa dimasup lagi karena lokasi simpul tersebut dalam memori tidak diketahui lagi (kecuali apabila simpul ini ditunjuk oleh **pointer** yang lain). Operasi inilah yang disebut dengan operasi **mengkopi pointer**. Kita kembali lagi pada gambar 6.6 sebelumnya. Jika statemen yang kita berikan adalah:

t2^ := t1^;

Maka hasil yang kita peroleh adalah :



Gambar 6.8 dengan t2^ sama dengan t1^.

Operasi inilah yang disebut operasi **mengkopi isi simpul**. Dari ilustrasi di atas, jelaslah apa yang dimaksud dengan operasi **mengkopi pointer** dan **mengkopi isi simpul**. Sebagai ringkasan maka:

- jika dalam statemen pemberian tanda ^ tidak ditulis, operasinya disebut operasi **mengkopi pointer**, dengan konsekuensi simpul yang semula ditunjuk oleh suatu **pointer** akan bisa terlepas dan tidak bisa dimasup lagi.
- jika dalam statemen pemberian tanda ^ ditulis, operasinya disebut operasi **mengkopi isi simpul pointer**; dengan konsekuensi bahwa isi dua simpul atau lebih akan menjadi sama.

Perlu kita ingat bahwa statemen pemberian hanya bisa dilaksanakan untuk perubah-perubah yang bertipe sama. Dengan demikian, statemen-statemen berikut ini adalah tidak benar.

```
t1 := t2^;
t2^ := t1;
```

Selain kedua operasi di atas, ada operasi lain dengan menggunakan relasi (<, <=, >=, >, <>). Untuk **pointer**, karena hanya berurusan dengan alamat tertentu dalam memori, maka hanya bisa dioperasikan dengan dua operator yaitu = atau <>. Sebagai contoh, misalnya:

```
If      t1 = t2  then
If      t1 = nil then
While  t2 <> nil do
```

Perhatikan bahwa kondisi t1 = t2 menanyakan apakah kedua **pointer** ini menunjuk ke lokasi yang sama. Untuk simpul, maka semua operator relasi bisa digunakan kondisi-kondisi seperti:

```
If      t1^ = t2^  then
While          t1^  <> t2^  do
```

Akan membandingkan isi simpul yang ditunjuk oleh **pointer** t1 dan t2.

6 Menghapus Pointer

Di atas telah dijelaskan bahwa **pointer** yang telah dialokasikan (dibentuk) bisa didealokasikan (dihapus) kembali pada saat program dieksekusi. Setelah suatu **pointer** dihapus, maka lokasi yang semula ditempati oleh simpul yang ditunjuk oleh **pointer** tersebut akan bebas, sehingga bisa digunakan oleh perubah lain. Statemen untuk menghapus **pointer** adalah **dispose**, yang mempunyai bentuk umum:

```
Dispose (perubah)
```

Dengan perubah adalah sembarang perubah yang bertipe **pointer**. Sebagi contoh, dengan menggunakan deklarasi:

```
Type
        Mendaftar = (ya, tidak);
        Tanggal    = record
                        Bulan           : 1..12;
                        Tahun           : 0..99;
                     End;
        Siswa    record
                        Nomor_mhs       : string[10];
                        Nama_mhs        : string[30];
                        Tgl_daftar      : mendaftar of
                        Ya              : (jum_mk     : integer;
                                           Jum_sks    : integer;
                                           Ip_sem     : real;
                        Tidak           : (tgl        : tanggal;
                                           Alasan     : string[15]);
                   End;
        Daf_siswa   = ^siswa;
Var
        Murid, murid1 : daf_siswa;
```

Kemudian kita membentuk simpul baru, yaitu:

```
New (murid);
Murid1 := murid;
```

Pada suatu saat, simpul yang ditunjuk oleh **pointer** murid1 tidak digunakan lagi, maka bisa dihapus dengan menggunakan statemen:

```
Dispose (murid);
```

Demikian penjelasan tentang perubah dinamis yang lebih dikenal dengan sebutan **pointer**. **Pointer** bisa digunakan untuk menyajikan struktur-struktur **tumpukan**, **antrian**, **senarai berantai**, dan **pohon**. Dalam beberapa contoh juga dijelaskan bagaimana mengurutkan data yang disajikan menggunakan **pointer**.

6.4 Contoh program pointer

```
program deklarasi_pointer;
uses
   crt;
type
   str30 = string[30];
   point = ^data;
   data  = record
             nama_mhs : str30;
             alamat   : str30;
         end;
var
```

```
  p1, p2 : point;
  a, b, c : str30;
begin
    clrscr;
    new(p1);
    new(p2);
    p1^.nama_mhs := 'Budi';
    p1^.alamat := 'Yogyakarta';
    writeln('Nama Mahasiswa Pertama: ',p1^.nama_mhs);
    writeln('Alamatnya adalah ',p1^.alamat);
    writeln;
    writeln;
    writeln('Setelah pointer kedua disamakan dengan pointer pertama');
    writeln('yakni, p2^ := P1^, berarti isi pointer pertama dikopi ke');
    writeln('pointer kedua');
    writeln('jadi Pointer kedua adalah : ');
    writeln;
    writeln;
    p2^:=p1^;
    writeln('Nama Mahasiswa kedua: ',p2^.nama_mhs);
    writeln('Alamatnya adalah ',p2^.alamat);
    readln;
end.
```

==========

Hasil program sebagai berikut:

==========

```
Nama Mahasiswa Pertama: Budi
Alamatnya adalah Yogyakarta


Setelah pointer kedua disamakan dengan pointer pertama
yakni, p2^ := P1^, berarti isi pointer pertama dikopi ke
pointer kedua
jadi Pointer kedua adalah :


Nama Mahasiswa kedua: Budi
Alamatnya adalah Aceh utara
Contoh program pointer
```

==========

```
Program contoh_pointer;
uses
   crt;
```

```
type
   pointer=^List;
List=record
        nama:string[25];
        nim:string[10];
        alamat:string[30];
        ipk:integer;
        next:pointer;
    end;
var
  awal,baru,r:pointer;
  j:integer;
begin
    clrscr;
    new(baru);
    write('Masukkan Nama   : ');
    readln(baru^.Nama);
    write('Masukkan Nomor Mahasiswa: ');
    readln(baru^.nim);
    write('Masukkan Alamat   : ');
    readln(baru^.alamat);
    write('IPK   : ');
    readln(baru^.ipk);
    awal:=baru;
    r:=baru;
    for j:=2 to 3 do
    begin
        new(baru);
        write('Masukkan Nama   : ');
        readln(baru^.Nama);
        write('Masukkan Nomor Mahasiswa : ');
        readln(baru^.nim);
        write('Masukkan Alamat   : ');
        readln(baru^.alamat);
        write('IPK   : ');
        readln(baru^.ipk);
        r^.next:=baru;
        r:=r^.next;
    end;
    baru^.next:=nil;
    r:=awal;
    for j:=1 to 3 do
    begin
        writeln('Nama   : ',r^.Nama);
        writeln('Nomor Mahasiswa: ',r^.nim);
        writeln('Alamat   : ',r^.alamat);
        writeln('IPK   : ',r^.ipk);
        r:=r^.next;
```

```
    end;
    readln;
end.
```

==========

Outpunya sebagai berikut:

```
Masukkan Nama      : salman
Masukkan Nomor Mahasiswa: 98/743/ps
Masukkan Alamat    : jl. solo km 5,2 no. 27
IPK        : 60
Masukkan Nama      : ali
Masukkan Nomor Mahasiswa : 97/564/ps
Masukkan Alamat    : jl. gajayan
IPK        : 80
Masukkan Nama      : budi
Masukkan Nomor Mahasiswa : 00/546/ps
Masukkan Alamat    : jl. bonjo1
IPK        :  98
Nama      : salman
Nomor Mahasiswa: 98/743/ps
Alamat    : jl. solo km 5,2 no. 27
IPK        :  60
Nama      : ali
Nomor Mahasiswa: 97/564/ps
Alamat    : jl. gajayana
IPK        :  80
Nama      : budi
Nomor Mahasiswa: 00/546/ps
Alamat    : jl. bonjol
IPK        :  98
```

This page is intentionallly left blank

BAB 7
PENGURUTAN (SORTING)

PENDAHULUAN

Deskripsi Singkat

Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi pengurutan, pengurutan larik, metode pengurutan dan disertai dengan contoh program pengurutan

Tujuan Instruksional Khusus

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi pengurutan, pengurutan larik, metode pengurutan dan mampu membuat program pengurutan.

PENYAJIAN

7.1 Pengertian Pengurutan

Pengurutan data (*sorting*) (ada juga yang menyebutnya sebagai pemilahan data) secara umum bisa didefinisikan sebagai suatu proses untuk menyusun kembali himpunan obyek menggunakan aturan tertentu. Secara umum ada dua jenis pengurutan data, yaitu pengurutan secara urut naik (*ascending*), yaitu dari data yang nilainya paling kecil sampai data yang nilainya paling besar; atau pengurutan data secara urut turun (*descending*), yaitu dari data yang mempunyai nilai paling besar sampai paling kecil. Dalam hal pengurutan data yang bertipe **string** atau **char**, nilai data dikatakan lebih kecil atau lebih besar dari yang lain didasarkan pada urutan relatif (*collating sequence*) seperti dinyatakan dalam tabel ASCII. Tujuan pengurutan data adalah untuk lebih mempermudah pencarian data dikelak kemudian hari.

Pengurutan data menjadi satu bagian yang penting dalam ilmu komputer karena waktu yang diperlukan untuk melakukan proses pengurutan perlu dipertimbangkan. Selain itu, masih ada beberapa aspek lain yang cukup menarik untuk dipelajari. Data yang harus kita urutkan tentunya sangat bervariasi baik dalam hal banyak data maupun jenis data yang akan diurutkan. Sayangnya, tidak ada satu algoritma yang terbaik untuk setiap situasi yang kita hadapi. Bahkan cukup sulit untuk menentukan algoritma mana yang paling baik untuk situasi tertentu karena ada beberapa faktor yang mempengaruhi efektifitas algoritma pengurutan. Beberapa faktor yang mempengaruhi pada efektifitas suatu algoritma pengurutan antara lain: banyak data yang akan diurutkan, kapasitas pengingat apakah mampu menyimpan semua data yang kita miliki, tempat penyimpanan data. Keuntungan yang bisa kita peroleh dari data yang sudah dalam keadaan terurutkan antara lain adalah bahwa data mudah dicari (misalnya dalam buku telepon atau kamus bahasa), mudah untuk dibetulkan, dihapus, disisipi, atau digabungkan. Dalam keadaan terurutkan, kita mudah mencek apakah ada data yang hilang (misalnya dalam tumpukan kartu bridge). Pengurutan juga digunakan dalam mengkompilasi program komputer jika tabel-tabel simbol harus dibentuk, dan juga memegang peran penting untuk mempercepat proses pencarian data yang harus dilakukan berulang kali. Seperti dijelaskan di atas pemilihan algoritma sangat ditentukan oleh struktur data yang digunakan. Dengan alasan ini maka sejumlah metoda pengurutan yang akan dijelaskan bisa diklasifikasikan menjadi dua kategori, yaitu pengurutan larik (*array*), dan pengurutan berkas masup urut (*sequential access file*). Kategori yang pertama disebut sebagai pengurutan secara internal, dan kategori kedua disebut dengan pengurutan eksternal. Dikatakan demikian karena larik tersimpan dalam memori utama komputer yang mempunyai kecepatan tinggi, sedangkan berkas biasanya tersimpan dalam pengingat luar (tambahan), misalnya cakram, atau disk. Contoh sederhana untuk membedakan dua kategori ini adalah pengurutan sejumlah kartu yang telah diberi nomor. Penyusunan kartu sebagai sebuah larik bisa kita bayangkan bahwa semua kartu terletak dihadapan kita sehingga semua kartu terlihat dengan jelas nomornya, dan setiap kartu bisa dimasup sendiri-sendiri. Penyusunan kartu sebagai sebuah berkas, agak berbeda, yakni semua kartu kita tumpuk sehingga hanya kartu bagian atas saja yang bisa kita lihat nomornya. Batasan ini akan membawa konsekuensi yang serius dalam pemilihan metoda yang akan digunakan, tetapi mungkin tidak bisa dihindarkan karena jumlah kartu cukup banyak sehingga meja tidak cukup luas untuk meletakkan seluruh kartu di atasnya.

7.2 Pengurutan Larik

Dalam pengurutan larik, yang disimpan dalam pengingat utama komputer, ada aspek ekonomis yang perlu dipertimbangkan. Aspek ini antara lain menyangkut kapasitas pengingat yang tersedia. Aspek lain adalah dalam hal waktu, yaitu waktu yang diperlukan untuk melakukan permutasi sehingga semua elemen akhirnya menjadi terurutkan. Ukuran efisiensi yang baik bisa diperoleh dari banyaknya pembandingan dan perpindahan yang harus dilakukan. Angka-angka ini merupakan fungsi dari N yaitu banyaknya elemen yang akan diurutkan. Algoritma yang baik memerlukan pembandingan sebanyak N log N kali. Meskipun demikian kita akan melihat beberapa metoda yang disebut dengan metoda langsung (*straight method*), yang seluruhnya memerlukan N^2 pembandingan. Metoda langsung ini bisa

dikelompokkan menjadi tiga metoda, yaitu penyisipan (*insertion*), seleksi (*selection*), dan penukaran (*exchange*). Sebelum kita melihat masing-masing metoda, marilah kita melihat terlebih dahulu deklarasi larik yang akan kita gunakan. Dalam hal ini kita akan menggunakan larik dimensi satu (vektor) yang elemennya bertipe **real**. Anda bisa mengganti tipe elemen ini dengan tipe data yang lain. Deklarasinya adalah:

```
const N = 100;
type Larik = array [1 .. N] of real;
```

Dalam deklarasi di atas, N adalah banyaknya elemen vektor. Anda bisa mengubah nilai konstanta N sesuai kebutuhan. Selain deklarasi di atas, berikut ini disajikan satu prosedur sederhana untuk menukar nilai dua buah elemen. Dalam prosedur ini, nilai elemen yang akan saling dipertukarkan antara perubah A dan B yang masing-masing bertipe **real**.

```
{*********************************************************
* Prosedur untuk menukarkan nilai dua buah elemen       *
*********************************************************}
procedure TUKARKAN (var A, B : real);
var T : real;

begin
  T := A;
  A := B;
  B := T
end;
```

Program 1. Prosedur untuk menukarkan nilai dua buah elemen

Prosedur TUKARKAN di atas akan digunakan oleh beberapa prosedur yang akan segera dijelaskan di bawah ini. Selain itu, metoda-metoda pengurutan masing-masing akan diimplementasikan dalam sebuah prosedur.

1. Metoda Penyisipan Langsung

Metoda penyisipan langsung (*straight insertion*) banyak digunakan oleh pemain kartu. Dalam metoda ini elemen-elemen (kartu) terbagi menjadi dua kelompok. Kelompok pertama, yaitu kelompok tujuan (kartu yuang ada di tangan pemain yang sudah dalam keadaan urut) dengan urutan $A_1 \ldots A_{I-1}$. Kelompok kedua disebut kelompok sumber (kartu yang masih ada di atas meja, dan masih belum terurutkan), dengan urutan $A_I \ldots A_N$. Dalam setiap langkah, dimulai dari I = 2, dengan pertambahan 1, elemen ke I diambil dari kelompok sumber akan dipindah ke kelompok tujuan dengan cara menyisipkannya pada tempatnya yang sesuai. Gambar 1 menyajikan contoh pengurutan menggunakan metoda penyisipan langsung pada sebuah vektor dengan 9 buah elemen.

Iterasi	A(0)*	A(1)	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)
I = 1	0	23	45	12	24	56	34	27	23	16
I = 2	45	23	45	12	24	56	34	27	23	16
I = 3	12	23	45	12	24	56	34	27	23	16
I = 4	12	12	23	45	24	56	34	27	23	16
I = 5	24	12	23	24	45	56	34	27	23	16
I = 6	56	12	23	24	45	56	34	27	23	16
I = 7	34	12	23	24	34	45	56	27	23	16
I = 8	27	12	23	24	27	34	45	56	23	16
I = 9	23	12	23	23	24	27	34	45	56	16
Akhir	16	12	16	23	23	24	27	34	45	56

[8] : Data sentinel

Gambar 7. 1. Ilustrasi metoda penyisipan langsung

Dari ilustrasi di atas bisa dilihat bahwa jika suatu elemen ke I akan disisipkan ke suatu tempat, misalnya pada posisi ke J, maka perlu dilakukan penggeseran ke kanan (ke subskrib yang lebih besar). Dalam hal ini perlu dihindarkan nilai J yang sama dengan 0 (dalam contoh di atas adalah pada langkah ketiga), yaitu dengan memberikan data sentinel, yang berupa A [0] = T, dengan T adalah nilai elemen ke I (yang akan disisipkan). Berikut adalah algoritma pengurutan dengan penyisipan langsung.

Algoritma SISIP_LANGSUNG

[Pengurutan elemen menggunakan metoda penyisipan langsung. Masukan dinyatakan sebagai vektor A (belum terurutkan), dan N (banyak elemen). Keluaran adalah vektor A yang sudah dalam keadaan terurutkan.]

Langkah 0 Baca vektor yang akan diurutkan (dalam program utama).
Langkah 1 Kerjakan langkah 2 sampai langkah 5 untuk I = 2 sampai N.
Langkah 2 Tentukan: T = A [I] (elemen yang akan disisipkan);
A [0] = T (data sentinel), dan
J = I – 1.
Langkah 3 (Lakukan penggeseran).
Kerjakan langkah 4 selama T < A [J].
Langkah 4 Tentukan: A [J + 1] = A [J], dan
= J – 1.
Langkah 5 Tentukan: A [J + 1] = T
Langkah 6 Selesai.

Prosedur yang mengimplementasikan algoritma di atas tersaji di bawah ini.

```
{*****************************************
* Prosedur untuk mengurutkan elemen vektor   *
* Dengan metoda PENYISIPAN LANGSUNG          *
*****************************************}
procedure SISIP_LANGSUNG (var A : Larik; N : integer);
var I,J : integer; T : real;

begin
  for I := 2 to N do
    begin
      T := A[I]; J := I – 1;
      A[0] := T;     {* data sentinel *}
  while T < A[J] do
    {* Selama elemen sebelah kiri data yang *
     * ditinjau lebih kecil dari data yang  *
     * akan disisipkan, geser ke kiri       *}
        begin
          A[J+1] := A[J];
          Dec (J)
    end;
      A[J+1] := T   {* menempatkan elemen *}
   end
end;
```

Program 2. Pengurutan vektor dengan metoda penyisipan langsung

Dalam prosedur di atas, banyak pembandingan (C) untuk nilai I tertentu adalah sebanyak I-1 kali, dengan paling sedikit satu kali, dan rata-rata sebesar I/2 kali. Banyaknya pemindahan atau penggeseran (M) untuk nilai I tertentu adalah C_I+2 (termasuk sentinel). Dengan demikian, banyaknya pembandingan dan pemindahan adalah:

$$C_{min} = N - 1 \qquad M_{min} = 2\,(N - 1)$$
$$C_{rata2} = (N^2 + N + 2)/4 \qquad M_{rata2} = (N^2 + 9N - 10)/4$$
$$C_{max} = (N^2 + N - 2) \qquad M_{max} = (N^2 + 3N - 4)/2$$

Banyaknya pembandingan dan pemindahan akan minimum jika elemen-elemennya semua sudah dalam keadaan urut; akan maksimum jika elemen-elemennya semua sudah dalam keadaan urut tetapi secara urut turun (tujuan kita adalah urut naik).

2. Penyisipan Biner

Dengan cara penyisipan langsung, pembandingan selalu dimulai dari elemen pertama, sehingga untuk menyisipkan elemen ke I kita harus melakukan pembandingan sebanyak I-1 kali. Hal ini bisa dipercepat dengan mengingat bahwa elemen pertama sampai elemen ke I-1 telah dalam keadaan urut, sehingga pembandingan tidak harus dimulai dari elemen pertama. Metoda penyisipan biner (*binary insertion*) merupakan perbaikan terhadap metoda penyisipan langsung, dimana pembandingan tidak dilakukan dimulai dari elemen pertama sampai ke I-1, tetapi dengan pembandingan biner. Caranya, kelompok yang sudah dalam keadaan urut (elemen ke 1 sampai I-1) dibagi menjadi dua bagian, kemudian elemen yang akan disisipkan dilihat kira-kira akan disisipkan pada bagian pertama atau kedua. Jika diperkirakan akan menempati bagian pertama, maka bagian kedua tidak perlu dilihat lagi, atau sebaliknya. Cara ini diulang sampai posisi yang tepat untuk elemen baru ditemukan. Sebagai contoh, kita ambil satu langkah pada proses penyisipan langsung, yaitu dalam langkah ke-7, dengan akan disisipkannya elemen yang bernilai 27.

I = 6	12	23	24	34	45	56	27	23	26
I = 7	12	23	24	27	34	45	56	23	26

Pada langkah ke-7 kita telah memperoleh 6 elemen pertama dalam keadaan terurutkan. Keenam elemen ini kita bagi dua (sama besar), yaitu (12 23 24) dan (34 45 56). Karena yang akan disisipkan adalah 27, maka pastilah akan menempati bagian (34 45 56). Dengan demikian pencarian posisi dilakukan hanya pada bagian ini. Bagian ini dipecah lagi menjadi dua bagian dan dicari lagi sampai posisi nilai yang akan disisipkan diketahui dengan tepat. Berikut adalah algoritmanya.

Algoritma SISIP_BINER

Pengurutan elemen menggunakan metoda penyisipan biner. Masukan dinyatakan sebagai vektor A (belum terurutkan), dan N (banyak elemen). Keluaran adalah vektor A yang sudah dalam keadaan terurutkan.

Langkah 0 Baca vektor yang akan diurutkan (dalam program utama).
Langkah 1 Kerjakan langkah 2 sampai langkah 8 untuk I = 2 sampai N.
Langkah 2 Tentukan: T = A [I] (elemen yang akan disisipkan);
Kiri = 1 (batas kiri), dan
Kanan = I – 1 (batas kanan).
Langkah 3 (Lakukan pembandingan biner).
Kerjakan langkah 4 dan 5 selama Kiri <= Kanan.
Langkah 4 (Membagi elemen pertama sampai I-1 menjadi dua bagian.)
Tentukan: Tengah = (Kiri + Kanan) div 2.
Langkah 5 Test: apakah T < A[Tengah]?
Jika ya, tentukan: Kanan = Tengah-1.
Jika tidak, tentukan: Kiri = Tengah+1.
Langkah 6 (Lakukan penggeseran.)
Kerjakan langkah 7 untuk J = I-1 sampai dengan Kiri dengan langkah mundur.
Langkah 7 Tentukan: A[J+1] = A[J].

Langkah 8 Tentukan: A[Kiri] = T.
Langkah 9 Selesai.

Prosedur yang mengimplementasikan algoritma di atas tersaji di bawah ini.

```
{*****************************************
* Prosedur untuk mengurutkan elemen vektor   *
* Dengan metoda PENYISIPAN BINER             *
*****************************************}
procedure SISIP_BINER (var A : Larik; N : integer);
var I,J,Kiri,Kanan,Tengah : integer;
   T ; real;

Begin
  for I := 2 to N do
    begin
      {* Elemen yang akan dipasang *}
     T := A[I];

     {* Batas pencarian *}
     Kiri := 1;
     Kanan := I – 1;
     while Kiri <= Kanan do
    {* Pencarian biner *}
        begin
          Tengah := (Kiri + Kanan) div 2;
          if T < A[Tengah] then
            {* Elemen baru di bagian pertama *}
            Kanan := Tengah-1
          Else
            {* Elemen baru di bagian kedua *}
            Kiri := Tengah+1
    end;

    {* Menggeser *}
    for J := I – 1 down to Kiri do
       A[J+1] := A[J];

    {* Menempatkan elemen baru pada posisinya *}
    A[Kiri] := T {* menempatkan elemen baru *}
  End
end;
```

Program 3. Pengurutan vektor dengan metoda penyisipan biner.

Telah dikatakan di atas bahwa metoda penyisipan biner adalah perbaikan dari metoda penyisipan langsung. Dalam hal ini yang diperbaiki adalah banyaknya pembandingan, sedangkan banyaknya penggeseran tetap. Dalam metoda ini banyaknya pembandingan adalah:

$$C = (\log N - \log e \pm 0.5)$$

3. Metode Seleksi

Cara kerja metode seleksi didasarkan pada pencarian elemen dengan nilai terkecil, kemudian dilakukan penukaran dengan elemen ke – I. Secara singkat, metoda ini bisa dijelaskan sebagai berikut. Pada langkah pertama, dicari data yang terkecil dari data pertama sampai data terakhir. Kemudian data terkecil tersebut kita tukar dengan data pertama. Dengan demikian, data pertama sekarang mempunyai nilai paling kecil dibandingkan data yang lain. Pada langkah kedua, data terkecil kita cari mulai data kedua sampai data terakhir. Data terkecil yang kita peroleh ditukar dengan data kedua. Demikian seterusnya sampai seluruh vektor dalam keadaan terurutkan. Untuk lebih memperjelas proses pengurutan dengan metoda seleksi, berikut disajikan contoh metoda ini.

Iterasi	A(1)*	A(2)	A(3)	A(4)	A(5)	A(6)	A(7)	A(8)	A(9)
I = 1, Lok=3	23	45	12	24	56	34	27	23	16
I = 2, lok=9	12	45	23	24	56	34	27	23	16
I = 3, lok=3	12	16	23	24	56	34	27	23	45
I = 4, lok=8	12	16	23	24	56	34	27	23	45
I = 5, lok=8	12	16	23	23	56	34	27	24	45
I = 6, lok=7	12	16	23	23	24	34	27	56	45
I = 7, lok=7	12	16	23	23	24	27	34	56	45
I = 8, lok=9	12	16	23	23	24	27	34	56	45
Akhir	12	16	23	23	24	27	34	45	45

Gambar 7.2. ilustrasi pengurutan dengan metoda seleksi.

Berdasarkan ilustrasi di atas, kita susun algoritmanya sebagai berikut:

Algoritma SELEKSI

Pengurutan elemen menggunakan metoda seleksi. Masukan dinyatakan sebagai vektor A (belum terurutkan), dan N (banyak elemen). Keluaran adalah vektor A yang sudah dalam keadaan terurutkan.

Langkah 0 Baca vektor yang akan diurutkan (dalam program utama).
Langkah 1 Kerjakan langkah 2 sampai langkah 4 untuk I = 1 sampai N-1.
Langkah 2 Tentukan: lok = I
Kerjakan langkah 3 untuk J = I + 1 sampai N.
Langkah 3 (Mencari data terkecil).
Test: apakah A[lok] > A[j] ?
Jika ya, tentukan: lok = J.

Langkah 4 Tukarkan nilai A[lok] dengan A[I]
Langkah 5 Selesai.

Berdasarkan algoritma di atas kita susun prosedurnya seperti tersaji dalam program 4.

```
{*****************************************
* Prosedur untuk mengurutkan elemen vektor   *
* Dengan metoda SELEKSI                      *
*****************************************}
procedure SELEKSI (var A : Larik; N : integer);
var
I,J,lok : integer;


Begin
  for I := 1 to N-1 do
    begin
      { lokasi elemen terkecil }
     lok := I;

      { Mencari elemen terkecil dan mencatat posisinya}
       for J := I + 1 to N do
    if A[lok] > A[J] then
    {lokasi elemen terkecil yang baru }
    lok := J;

    { Menukar elemen pada lokasi ke I  }
     dengan elemen pada lokasi ke lok }
     TUKARKAN (a[I], A[lok])
  End
end;
```

Program 4. Pengurutan vektor dengan metode seleksi.

Metoda ini secara garis besar merupakan kebalikan dari metoda penyisipan langsung. Dalam setiap langkah pada metoda penyisipan langsung kita hanya memperhatikan satu elemen dari sumber dan semua elemen dari larik tujuan untuk menetukan posisinya yang tepat; sehingga sering disebut dengan *one source-multiple destinations*. Dalam metoda ini seleksi terjadi sebaliknya, yakni kita memperhatikan semua elemen dalam larik sumber untuk menentukan elemen terkecil yang akan kita tempatkan pada tujuan; sehingga sering disebut dengan *multiple source one destination.* Dalam metoda ini banyaknya pembandingkan (untuk mencari elemen dengan nilai terkecil) besarnya tak gayut terhadap urutan semula. Banyaknya pembandingkan adalah sebesar:

$$C = (N^2 - N) / 2$$

4. Metode Gelobang

Metoda gelombang (*BubleSort*), sering juga disebut dengan metoda penukaran (*ExchangeSort*), adalah metoda yang mendasarkan penukaran dua buah elemen untuk mencapai keadaan ururt yang diingkinkan. Metoda ini cukup mudah untuk dipahami dan diprogram. Tetapi dari beberapa metoda yang akan pelajari, metoda ini merupakan metoda yang paling tidak efisien.

Iterasi ke	A(1)*	A(2)	A(3)	A(4)	A(5)
Awal	24	23	56	45	12
I = 1	23	24	56	45	12
	23	24	56	45	12
	23	24	45	56	12
	23	24	45	12	56
I = 2	23	24	45	12	56
	23	24	45	12	56
	23	24	12	45	56
I = 3	23	24	12	45	56
	23	12	24	45	56
I = 4	12	23	24	45	56
Akhir	12	23	24	45	56

Gambar 7.3. Ilustrasi metoda gelombang.

Untuk membawa vektor menjadi dalam keadaan urut bisa dilaksanakan dengan dua cara. Cara pertama adalah selalu meletakkan elemen dengan nilai paling besar pada posisi terakhir (posisi ke N). kemudian elemen dengan nilai paling besar kedua diletakkan pada posisi N-1, dan seterusnya. Cara kedua adalah kebalikan cara pertama. Dalam hal ini yang digunakan sebagai patokan adalah nilai terkecil. Dengan kata lain, pada iterasi pertama kita akan meletakkan elemen dengan nilai terkecil pada posisi ke 1. Kemudian elemen dengan nilai elemen terkecil kedua diletakkan pada posisi ke 2, dan seterusnya. Dalam hal ini kita akan menggunakan cara pertama. Gambar 3 menunjukkan pelacakan metoda gelombang untuk mengurutkan vektor dengan 5 elemen. Dari ilustrasi di atas kita bisa melihat, bahwa untuk vektor dengan N elemen akan memerlukan iterasi sebanyak N-1 kali, di samping itu bisa kita perhatikan pula bahwa apabila nomor iterasi kita tambah dengan banyaknya langkah pada iterasi tersebut, besarnya selalu tetap, yaitu sama dengan N. sebagai contoh, untuk iterasi ke 1 terdapat 4 langkah; iterasi kedua terdapat 3 langkah, dan seterusnya. Berikut disajikan algoritma Gelombang yang meringkas proses di atas.

Algoritma GELOMBANG

Pengurutan elemen menggunakan metoda gelombang. Masukan dinyatakan sebagai vektor A (belum terurutkan), dan N (banyak elemen). Keluaran adalah vektor A yang sudah dalam keadaan terurutkan.

Langkah 0 Baca vektor yang akan diurutkan (dalam program utama).
Langkah 1 Kerjakan langkah 2 untuk I = 1 sampai N-1.
Langkah 2 Kerjakan langkah 3 untuk J = 1 sampai N-I.

Langkah 3 Test: apakah A[J] > A[J+1] ?
Jika ya, tukarkan nilai kedua elemen ini.
Langkah 5 Selesai.

Berdasarkan algoritma di atas, kita bisa menyusun prosedurnya seperti tersaji dalam program 5. Di atas telah disebutkan bahwa metoda ini merupakan metoda yang paling tidak efesien. Alasanya adalah bahwa apabila kita mengurutkan vektor sebanyak N elemen dan pada iterasi yang kurang dari N-1, maka iterasi tersebut harus tetap dilaksanakan sampai N-1. Dengan demikia, dalam metoda Gelombang akan terjadi pembandingkan dan pemindahan atau penukaran dua elemen sebanyak:

$$C = (N^2 - N) / 2$$

$$M_{min} = 0$$
$$M_{rata2} = 3\ (N^2 - N) / 4$$
$$M_{maks} = 3\ (N^2 - 4) / 2$$

```
{prosedur untuk mengurutkan elemen vektor dengan metoda
gelombang BUBLESORT}
procedure BUBLE_SORT (var A : larik; N : integer);
var
    i, j : integer;
begin
      for i := 1 to n-1 do
          for j := 1 to n-1 do
              if a[j] > a[j+1] then
                  tukarkan (a[j], a[j+1])
end;
```

Program 5. Pengurutan vektor menggunakan metoda gelombang (metoda penukaran).

7.3 Contoh Sebuah program sorting dengan metoda bubble sort:

```
program urut;
uses
   crt;
const
    maks = 10;
type
     arr = array [1..maks] of byte;
var
  i : byte;
  data : arr;
procedure masukkan;
begin
    clrscr;
    writeln('Masukkan data sebanyak 10 : ');
```

```
    writeln('------------------------------');
    for i := 1 to maks do    {masukkan 10 data}
    begin
        writeln('Data ke- ',i,' = ');
        readln(data[i]);
    end;
    clrscr;
    for i := 1 to maks do  {mencetak data}
    write(data[i],'  ');
    writeln;
    writeln('----------------------------');
    writeln('Data yang telah diurutkan : ');
end;
procedure tukar (var a, b : byte);       {prosedur untuk menukar data}
var
   c : byte;
begin
    c:=a;
    a:=b;
    b:=c;
end;
procedure asc; { pengurutan secara ascending }
var
  p, q : byte;
  flag : boolean;
begin
    flag := false;
    p:=2;
    while (p < maks) and (not flag) do
    begin
        flag := true;
        for q := maks downto p do
        if data[q] < data [q-1] then
        begin
            tukar(data[q], data[q-1]);
            flag := false;
        end;
        inc(i);
    end;
write('Ascending : ');
end;
procedure keluaran;
begin
    for i := 1 to maks do
    write(data[i], ' ');
    writeln;
end;
begin    { program utama }
```

```
        masukkan;
        asc;
        keluaran;
        readkey;
end.
```

Hasilnya sebgai berikut:
30 20 34 56 1 2 34 56 78 20

Data yang telah diurutkan :
Ascending : 1 2 20 20 30 34 34 56 56 78

PENUTUP

Tugas

1. Carilah 2 metode sorting lainya dan tuliskan dalam paper (makalah) beserta Programnya, Algoritma, Flowchart dan analisis tiap-tiap metode sorting yang ada.

2. Dengan Menggunakan Algoritma (langkah-langkah) yang ada pada Pengurutan data metode seleksi, Buatlah:
   a. Flowchart System
   b. Program untuk Pengurutan data metode seleksi

This page is intentionallly left blank

BAB 8
PENCARIAN (SEARCHING)

PENDAHULUAN

Deskripsi Singkat
Perkuliahan akan dimulai dengan pembahasan materi yang meliputi definisi pencarian, pencarian berurutan, pencarian biner dan disertai dengan contoh program pencarian.

Tujuan Instruksional Khusus
Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi pencarian, pencarian berurutan, pencarian biner dan mampu membuat program pencarian.

PENYAJIAN

8.1 Pengertian Pencarian

Pencarian data yang sering juga disebut dengan *table look-up* atau *storage and retrieval information*, adalah suatu proses untuk mengumpulkan sejumlah informasi didalam pengingat computer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin. Seperti pada pengurutan data, metode-metode pencarian data juga bisa dikelompokan dengan beberapa cara. Cara pertama adalah dengan mengelompokan metode pencarian kedalam **pencarian internal** (*internal searching*) dan **pencarian eksternal** (*external searching*). Dalam pencarian internal, semua rekaman yang diketahui berada dalam pengingat computer. Sedangkan dalam pencarian eksternal, tidak semua rekaman yang diketahui berada dalam pengingat computer, tetapi ada sejumlah rekaman yang tersimpan dalam penyimpanan luar, misalnya pita atau cakram magnetis. Cara pengelompokan kedua adalah dengan mengelompokannya menjadi **Pencarian Statis** dan **Pencarian Dinamis.** Dalam pencarian statis, banyaknya rekaman yang diketahui dianggap tetap. Sedangkan dalam pencarian dinamis banyaknya rekaman yang diketahui bisa berubah-ubah yang

disebabkan oleh penambahan atau penghapusan suatu rekaman. Pemilihan struktur data yang digunakan untuk menyimpan data yang diketahui akan mempengaruhi efisiensi pencarian itu sendiri. untuk itu, dalam bab ini penulis akan menerapkan beberapa metode untuk dua struktur data yang berbeda, yaitu menggunkan vector dengan deklarasi tipe data array dimensi satu, dan senarai berantai. Vector yang digunakan mempunyai deklarasi sebagai berikut:

```
Type Larik = array [1.. 1000] of integer;
```

Untuk senarai berantai, deklarasi simpul yang digunakan adalah:

```
Type List      = ^simpul;
     Simpul  = record
               Info     : integer;
               Kanan          : List;
     End;
```

Pada kedua deklarasi diatas, tipe informasi yang digunakan adalah integer; anda bisa menggunakan tipe data yang lain atau bahkan juga bisa ditambah dengan medan-medan yang lain. Jika dalam suatu program digunakan tipe data yang lain, secara tersendiri akan diberitahukan.

8.2 Pencarian Berurutan

Metode yang paling sederhana dari sejumlah metode pencarian adalah metode pencarian berurutan (sequential searching), secara garis besar metode ini bisa dijelaskan sebagai berikut; Dari vector yang diketahui, data yang dicari dibandingkan satu per satu sampai data tersebut ditemukan atau tidak ditemukan. Pada saat data yang dicari sudah ketemu, maka proses pencarian langsung dihentikan. Tetapi jika data yang dicari belum ketemu, maka pencarian diteruskan sampai seluruh data dibandingkan. Dalam kasus yang paling buruk, untuk vector dengan N elemen harus dilakukan pencarian sebanyak N kali pula. Dalam algoritma yang disajikan dibawah ini, jika data yang dicari tidak ditemukan, maka data tersebut akan ditambahkan pada vector yang sudah ada, dan diletakkan sebagai elemen paling akhir

Algoritma CARI_VEKTOR_URUT

Langkah 0 Baca vektor yang akan diketahui, misalnya sebagai vector A dengan N elemen.
Langkah 1 Baca data yang akan dicari, misalnya data.
Langkah 2 (Inisialisasi)
Tentukan: ada = False
Langkah 3 (Proses Pencarian).
Untuk I = 1 sampai N kerjakan langkah 4.
Langkah 4 Test apakah : Data = A [I] ?
Jika ya, (berarti data ketemu), tentukan:
Ada=True, posisi=I, dan I=N .
Langkah 5 (Menambah data pada vector, jika diperlukan)
Test apakah Ada=False?

Jika ya, (data tidak ketemu), tentukan:
N=N+1, dan A[I] = data.

Langkah 6 Selesai.

Prosedur dibawah ini menyajikan implementasi algoritma diatas yang diterapkan pada struktur data yang berbentuk Vektor

```
{*********************************************************
* Prosedur untuk mencari data pada suatu vektor  *
* Dengan metoda PENCARIAN BERURUTAN  *
*********************************************************}
procedure CARI_VEKTOR_URUT (var ada : Boolean;
                            var N,posisi : integer;
                            var A     : Larik;
                            data      : Integer);
var I : integer;

begin
     {*dianggap data tidak ada*}
     Ada := False;
     {*Iterasi Dimulai*}
     for I := 1 to N do
         if A[I] = data then
             {*data yang dicari ketemu*}
             begin
                posisi := I;
                 ada := true;
                 exit;              {*pencarian selesai*}

       if not ada then
          {*data yang dicari tidak ketemu*}
           *Sisipkan elemen kedalam vector*}
             Begin
                Inc(N);
                A[N] := data
             End;
   End.

     A[J+1] := T   {* menempatkan elemen *}
   end
end;
```

Prosedure Pencarian menggunakan metode pencarian berurutan pada sebuah vektor

8.3 Pencarian Biner

Cara kedua untuk mencari data pada vector yang elemennya telah diurutkan adalah menggunakan metode pencarian Biner (*Binary Search*). Jika kita bandingkan, maka metode ini akan jauh lebih cepat dibandingkan dengan metode pencarian berurutan. Metode pencarian biner dijelaskan sebagai berikut. Setelah vector yang diketahui diurutkan, vector tersebut dibagi menjadi dua sub vector yang mempunyai jumlah elemen yang sama. Kemudian data dibandingkan dengan data terakhir dari subvektor pertama. Jika data yang dicari lebih kecil, pencarian diteruskan pada sub vector pertama dengan terlebih dahulu membagi dua sub vector tersebut. Tetapi jika

data yang dicari lebih besar dari data terakhir pada sub vector pertama, berarti data yang dicari kemungkinan terletak pada sub vector kedua. Dengan demikian pencarian dilakukan pada sub vector kedua. Proses diatas diulang sampai data yang dicari ditemukan atau tidak ditemukan. Untuk lebih memperjelas penjelasan diatas perhatikan contoh berikut. Dimisalkan kita akan mencari data yang bernilai 20 pada vector berikut ini.

2	8	11	15	18	19	20	22	35	40	45

Vector diatas kita pecahkan menjadi 2 (dua) sub vector sebagai berikut:

2	8	11	15	18		19	20	22	35	40	45

Subvektor 1 Subvektor 2

Dari hasil pemecahan diatas kita lihat bahwa data yang bernilai 20 terdapat pada subvektor 2. dengan demikian pencarian kita laksanakan pada subvektor 2, subvektor 1 tidak perlu dihiraukan lagi. Subvektor 2 kemudian dipecah lagi menjadi:

19	20	22

subvektor 1

35	40	45

subvektor 2

Sekarang, data yang bernilai 20 terdapat pada subvektor 1, dengan demikian subvektor 1 kita pecah lagi. Proses diteruskan sampai data yang dicari ketemu atau tidak ketemu. Dari ilustrasi diatas kita susun algoritmanya sebagai berikut:

Algoritma BINER

Langkah 0 Baca vektor yang akan diketahui, misalnya vector A dengan N buah elemen, dan urutkan secara urut naik.
Langkah 1 Baca elemen yang akan dicari, misalnya data.
Langkah 2 (Inisialisasi)
Tentukan: ada = False,
Atas = N, dan
Bawah = 1.
Langkah 3 Kerjakan langkah 4 dan 5 selama Atas >= Bawah.
Langkah 4 (Menentukan batas subvektor)
Tentukan: Tengah = (Atas + Bawah) div 2
Langkah 5 Test nilai data terhadap A [tengah].
Jika data > A [tengah], (ada di subvektor 2),
tentukan: Bawah = tengah + 1.
Jika data < A [tengah], (ada di subvektor 1),
Tentukan: Atas = tengah – 1
Jika data = A [tengah], (data ketemu),
Tentukan: Ada = true,
Posisi = tengah, dan
Bawah = Atas + 1
Langkah 6 Selesai.

Algoritma pencarian biner yang dijelaskna diatas diimplementasikan dalam prosedur dibawah ini. Dalam hal ini, jika elemen yang akan dicari tidak ditemukan, program tidak akan menambahkanya sebagai elemen baru.

```
{*********************************************************
* Prosedur untuk mencari data pada suatu vector    *
* Dengan metoda PENCARIAN BINER                    *
*********************************************************}
procedure CARI_BINER (var ada : Boolean;
                      var N,posisi : integer;
                      var A     : Larik;
                      data      : Integer);
var Atas, Bawah, Tengah : integer;\
begin
     {*dianggap data tidak ada*}
     Ada := False;

     {*menentukan batas atas dan batas bawah
     Atas:= N;
     Bawah:= 1;

    {*Iterasi Dimulai*}
    While Atas >=  Bawah do
          Begin
            {*elemen yang ada ditengah*}
            Tengah : = (Atas + Bawah) div 2;

            If data < A [Tengah] then
                 {*data kemungkinan ada*
                  *di subvektor pertama*}
            Atas := Tengah – 1

            Else if data > A [Tengah] then
                 {*data kemungkinan ada*
                  *di subvektor kedua*}
            Bawah := Tengah + 1

            Else
                 {* data yang dicari sudah ada*}
                 Begin
                     Ada := true;
                     Posisi := Tengah;
                     Bawah := Atas + 1
                end;
            end;
end.
```

Prosedure Pencarian menggunakan metode pencarian Biner (Binary Search)

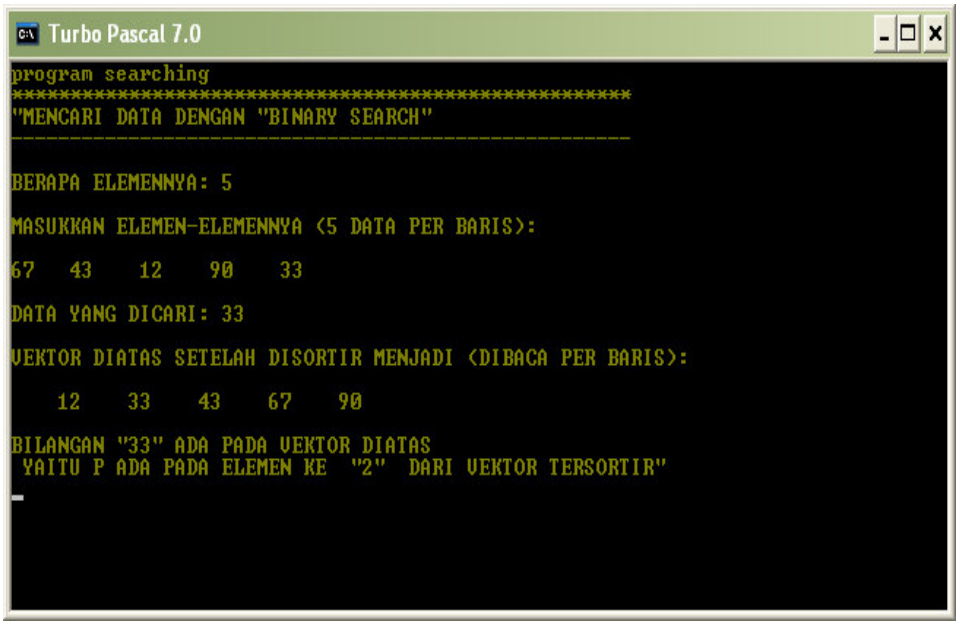
8.4 Contoh Program pencarian menggunakan metode pencarian Biner

```
Program menggunakan_konsep_searching;
uses crt;
type  Larik = array[1..100] of integer;
var  vektor  : Larik;
     Ketemu  : boolean;
     cari,
     Lokasi,
     Cacah,
     Baris   : integer;
{[**************************************]
 [ Prosedur baca elemen ]
 [**************************************]}
Procedure BACA_VEKTOR (var vektor : Larik; var N, Baris : integer);
var I, kolom : integer;

begin
  kolom := 1;
  write('BERAPA ELEMENNYA: ');readln(N);
  writeln;writeln('MASUKKAN ELEMEN-ELEMENNYA (5 DATA PER BARIS):
');
  writeln;
  for I := 1 to N do
      begin
        readln(vektor[I]);
        gotoXY((I mod 10)*6,whereY-1)
      end
end;

Procedure CETAK_VEKTOR (vektor : Larik; N : integer);
var   I : integer;

begin
  for I := 1 to N do
      begin
        write(vektor[I]:6);
   if I mod 10 = 0 then writeln;
      end
end;

Procedure SORTIR (var vektor : Larik; N : integer);
var  I, J, Posisi, Bantu    : integer;

begin
  for I := 1 to N-1 do
      begin
        posisi := I;
```

```
      for J := I+1 to N do
        if vektor[posisi] > vektor[J] then posisi := J;
      Bantu := Vektor[I];
      Vektor[I] := Vektor[posisi];
      Vektor[posisi] := Bantu;
    end
end;
procedure MENCARI_BINER (var ketemu : boolean; var Lokasi : integer;
                         var cari : integer;
                         vektor  : Larik; N : integer);
var Atas, bawah : integer;
begin
  writeln;writeln;
  write('DATA YANG DICARI: ');readln(Cari);
  Atas := N;
  Bawah := 1;
  Ketemu := false;
  while (not Ketemu) and (Bawah <= Atas) do
    begin
      Lokasi := (atas + Bawah) div 2;
      if cari = Vektor[Lokasi] then
    ketemu := true
      else if cari > vektor[lokasi] then
          bawah := lokasi + 1
      else
          atas := lokasi - 1
      end
end;
{[*****************]
 [ program utama ]
 [*****************]}
begin
textbackground(black);
  clrscr;
  textcolor(6);
  writeln('program searching');
  baris := 8;
  writeln('*******************************************************');
  writeln('"MENCARI DATA DENGAN "BINARY SEARCH"');
  writeln('------------------------------------------------------------');
  writeln;
  BACA_VEKTOR(vektor,cacah,baris);
  SORTIR(vektor,cacah);
  MENCARI_BINER(ketemu,lokasi,cari,vektor,cacah);
  writeln;write('VEKTOR DIATAS SETELAH DISORTIR ');
  WRITELN('MENJADI (DIBACA PER BARIS): '); WRITELN;
  CETAK_VEKTOR(vektor,cacah);  writeln;writeln;
  if ketemu then
```

```
    begin
      writeln('BILANGAN "',CARI:1,'" ADA PADA VEKTOR DIATAS');
      write(' YAITU P ADA PADA ELEMEN KE  "', LOKASI: 1 );
      writeln(  '"  DARI VEKTOR TERSORTIR" ');
    end
  else
    writeln('BILANGAN"',cari:1,'" TIDAK ADA PADA VEKTOR DIATAS');
    readln;
    end.
```

Hasil program pencarian



PENUTUP

Latihan:

1. Jika diketahui data sebagai berikut:
   4 9 13 14 20 23 28 29 41 54 61 63 69 74 80
   Data yang dicari = **69.**
   carilah data tersebut dengan menggunakan metode pencarian:
   a. Binary Search
   b. Sequential Search

TUGAS:

1. Carilah 2 metode searching (pencarian) lainya dan tuliskan dalam paper (makalah) beserta Programnya, Algoritma, Flowchart dan analisis tiap-tiap metode searching yang ada.
2. Dengan Menggunakan Algoritma (langkah-langkah) yang ada pada pencarian data, Buatlah:
   a. Flowchart metode sequential searching dan metode Binary searching
   b. Program dan outputnya untuk metode sequential searching

DAFTAR PUSTAKA

Ir. P. Insap santoso, M.Sc, Struktur data dengan Turbo Pascal Versi 6.0, Andi Offset Yogyakarta

D, Suryadi H.S,. Pengantar Struktur Data, Penerbit Gunadarma.

Loomis, Mary E.S,. Data Management and File Structures, Prentice Hall International Inc,. 1989.

Reynolds, W. Charles, Program Design and Data Structures in Pascal, Wadsworth Pub. Co,. 1986.

This page is intentionallly left blank

universitas
MALIKUSSALEH

Materi perkuliahan meliputi definisi dan bentuk umum Tipe data, seperti tipe data Array, Tipe data Record dan contoh program array dan record

Setelah materi ini diajarkan maka mahasiswa dapat menjelaskan definisi dan bentuk umum Tipe data, seperti tipe data Array, Tipe data Record dan contoh program array dan record.

Dalam bahasa pemrograman pascal, semua perubah yang akan dipakai harus sudah ditentukan tipe datanya. Dengan menentukan tipe data suatu perubah, sekaligus menentukan batasan nilai perubah tersebut dan jenis operasi yang bisa dilaksanakan atas perubah tersebut.

Bentuk umum dari deklarasi tipe data adalah:

Type pengenal = tipe;

Dengan, Pengenal : nama pengenal yang menyatakan tipe data.
Tipe : Tipe data yang digunakan.

Array adalah tipe terstruktur yang mempunyai komponen dalam jumlah yang tetap dan setiap komponen mempunyai tipe data yang sama. Posisi masing-masing komponen dalam array dinyatakan sebagai nomor index. Dalam bentuknya array dapat kita tinjau dari segi pengaturan struktur datanya dalam konteks dimensi sebagai berikut:

1. Array 1-Dimensi, contoh: List, Vektor
2. Array 2-Dimensi, contoh: Tabel, Matriks (2 dimensi)
3. Array 3-Dimensi, contoh: Matriks 3 dimensi

FAKULTAS TEKNIK
UNIVERSITAS MALIKUSSALEH