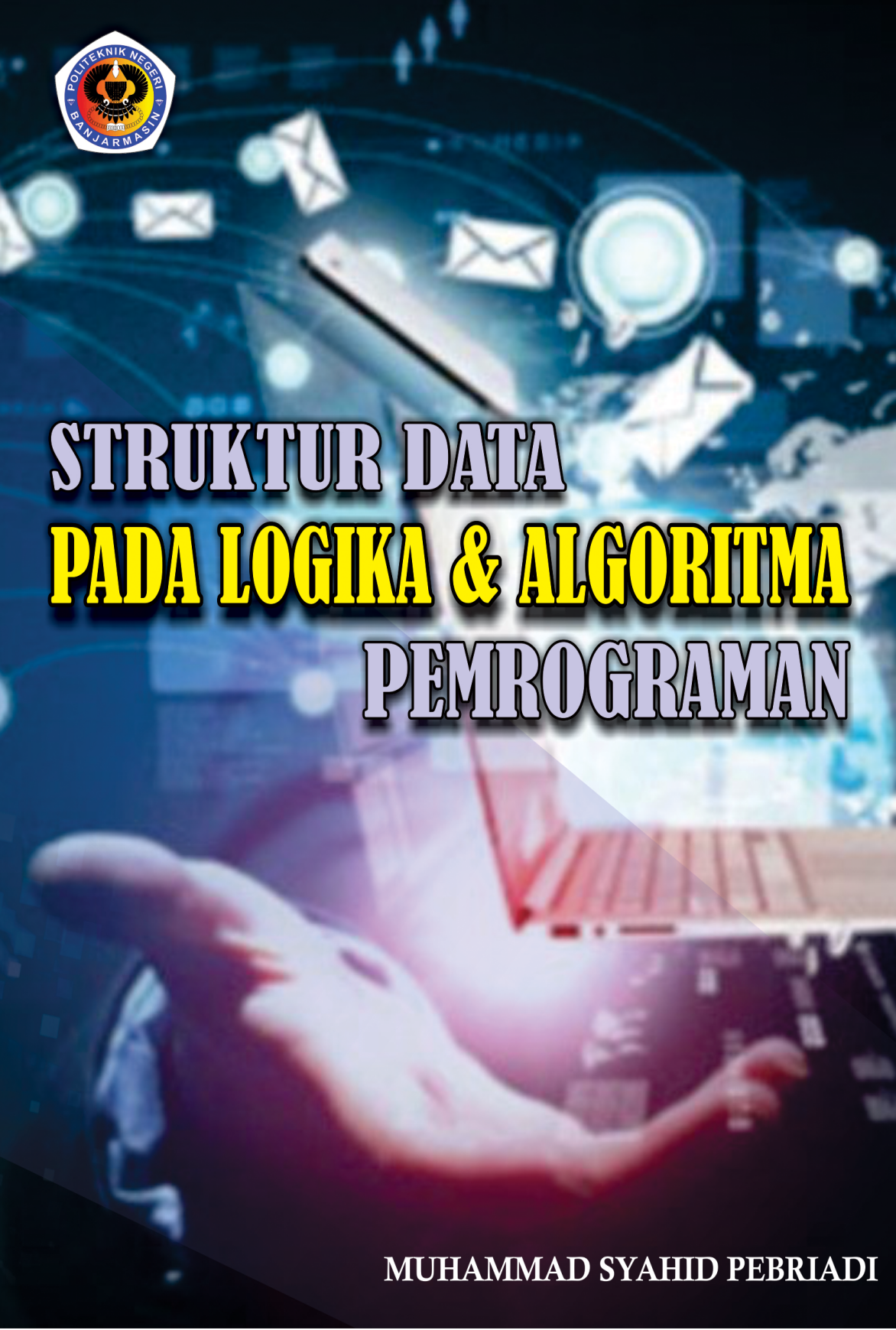


STRUKTUR DATA PADA LOGIKA & ALGORITMA PEMROGRAMAN

MUHAMMAD SYAHID PEBRIADI

STRUKTUR DATA PADA LOGIKA DAN ALGORITMA PEMROGRAMAN

Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Perlindungan Pasal 26

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap :

i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah).

STRUKTUR DATA PADA LOGIKA DAN ALGORITMA PEMROGRAMAN

M. Syahid Pebriadi

Poliban Press

STRUKTUR DATA PADA LOGIKA DAN ALGORITMA PEMROGRAMAN

Penulis :
M. Syahid Pebriadi

ISBN :
978-623-7694-84-7

ISBN Elektronik:
978-623-7694-85-4 (PDF)

Editor dan Penyunting :
Reza Fauzan

Desain Sampul dan Tata letak :
Eko Sabar Prihatin; Rahma Indera

Penerbit :
POLIBAN PRESS
Anggota APPTI (Asosiasi Penerbit Perguruan Tinggi Indonesia)
no.004.098.1.06.2019
Cetakan Pertama, 2022

Redaksi :
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry,
Pangeran, Komp. Kampus ULM, Banjarmasin Utara
Telp : (0511)3305052
Email : press@poliban.ac.id

Diterbitkan pertama kali oleh :
Poliban Press, Banjarmasin, Januari 2022

PRAKATA

Assalamu'alaikum wa rahmatullahi wa barakatuh.

Buku ini merupakan jenis buku teks mahasiswa yang menempuh mata kuliah Logika dan Algoritma Pemrograman. Mahasiswa yang menempuh kuliah bidang informatika dapat menggunakan buku ini sebagai sumber bacaan acuan maupun pelengkap.

Buku Struktur Data Pada Logika dan Algoritma Pemrograman terbagi menjadi 7 bab. Masing-masing bab dilengkapi dengan tujuan pembahasan, studi kasus, dan latihan soal pengayaan. Di dalam studi kasus juga terdapat beberapa soal latihan yang berkaitan dengan materi studi kasus, sehingga mahasiswa diharapkan memiliki kemampuan menganalisa yang lebih mendalam.

Mudah-mudahan, meskipun masih serba sederhana, bahan ajar ini bisa membantu mahasiswa untuk memahami beberapa aspek sistem struktur data secara kontekstual. Penulis yakin bahwa materi dalam bahan kuliah ini masih jauh dari kesempurnaan, sehingga terbuka untuk mendapatkan kritik dan saran untuk perbaikan pada semua sisi penulisannya.

Wassalamu'alaikum wa rahmatullahi wa barakatuh.

Banjarmasin, Agustus 2021
Penyusun

DAFTAR ISI

BAB I

PENDAHULUAN

Capaian Pembelajaran:

1. Mampu memahami konsep algoritma
2. Mampu menjelaskan bagian-bagian algoritma
3. Mampu menjelaskan alur pemrograman.

Algoritma berasal dari kata *Algoris* dan *Ritmis* yang pertama kali diungkapkan oleh Abu Ja'far Mohammed Ibn Musa Al-Khowarismi pada tahun 825 M dalam bukunya yang berjudul Al-Jabr Wa-al Muqabla.

Dalam bidang pemrograman, algoritma didefinisikan sebagai suatu metode khusus yang tepat dan terdiri dari serangkaian langkah yang terstruktur dan dituliskan secara sistematis yang akan dikerjakan untuk menyelsesaikan masalah dengan bantuan komputer.

Algoritma merupakan pola piker terstruktur yang berisi tahap-tahap penyelesian masalah yang dapat disajikan dengan teknik tulisan maupun dengan gambar. Penyajian algoritma dalam bentuk tulisan menggunakan Structure English atau Pseudo Code (Kode Semu). Sedangkan penyajian algoritma dalam bentuk gambar menggunakan flowchart (diagram alir).

Penilaian sebuah algoritma di dasarkan pada beberapa hal sebagai berikut:

a. Waktu Eksekusi
b. Penggunaan memori/ sumber daya
c. Kesederhanaan dan kejelasan algoritma

Waktu eksekusi sebuah algortima dipengaruhi oleh:

a. Jenis data input
b. Jumlah data input

c. Pemilihan instruksi bahasa pemrograman.

1.1. Dasar-Dasar Logika Informatika

Proposisi merupakan kalimat yang bias ditentukan nilai kebenarannya. Nilai kebenaran bias "benar" atau "salah", tetapi tidak keduanya sekaligus. Setiap proposisi adalah kalimat, tetapi setiap kalimat belum tentu sebuah proposisi.

Contoh Proposisi:

- 15 habis dibagi 3
- Malang adalah ibukota Indonesia

Contoh bukan Proposisi (kalimat terbuka)

- Belikan nasi bungkus di warung!
- Apakah hari ini hujan?

Jika P adalah suatu kalimat, demikian juga negasinya (Not P). Negasi dari P yang dinotasikan dengan ¬P adalah proposisi.

¬P = bukan P

Nilai kebenaran dari proposisi ¬P didefinisikan dengan tabel kebenaran berikut:

P	**¬P**
B	S
S	B

Misalkan P dan Q adalah proposisi, maka:

- Disjungsi P dan Q, dinotasikan sebagai P ∨ Q adalah Proposisi
- Konjungsi P dan Q, dinotasikan sebagai P ∧ Q adalah Proposisi

Proposisi hasil kombinasi dari proposisi –proposisi tersebut disebut kalimat majemuk.

Contoh:

- P = 2 + 5 =3
- Q = Satu minggu sama dengan 7 hari
- P ∨ Q = 2 + 5 =3 atau Satu minggu sama dengan 7 hari (nilai kebenaran = "benar")

- $P \wedge Q$ = 2 + 5 =3 dan Satu minggu sama dengan 7 hari (nilai kebenaran = ”salah”)

Nilai kebenaran dari proposisi $P \vee Q$ dan $P \wedge Q$ didefinisikan dengan tabel kebenaran berikut:

P	**Q**	**$P \wedge Q$**	**$P \vee Q$**
B	B	B	B
B	S	S	B
S	B	S	B
S	S	S	S

Proposisi Bersyarat

Jika P dan Q adalah Proposisi, maka Proposisi jika P maka Q disebut Proposisi bersyarat (Implikasi) dan dinotasikan sebagai $P \rightarrow Q$.

Proposisi P disebut Hipotesis (Anteseden) dan proposisi Q disebut konklusi.

Nilai kebenaran dari proposisi $P \rightarrow Q$ didefinisikan dengan tabel kebenaran berikut:

P	**Q**	**$P \rightarrow Q$**
B	B	B
B	S	S
S	B	B
S	S	B

Jika P dan Q adalah Proposisi, maka proposisi majemuk P jika hanya jika Q disebut Proposisi Bikondisional (Bi-implikasi) dan dinotasikan sebagai $P \leftrightarrow Q$.

Nilai kebenaran dari proposisi $P \leftrightarrow Q$ didefinisikan dengan tabel kebenaran berikut:

P	**Q**	**$P \leftrightarrow Q$**
B	B	B
B	S	S
S	B	S
S	S	B

Kesamaan Logika

Misalkan bahwa proposisi majemuk P dan Q terdiri dari proposisi P_1,P_2, ….., P_n, kita katakana P dan Q Ekivalen ($P \equiv Q$) yang menyatakan bahwa untuk nilai kebenaran sembarang yang diberikan dari P_1,P_2, ….., P_n, P dan Q keduanya bisa benar ataupun bisa salah.

1.2. Pseudocode

Pseudocode merupakan metode yang cukup efisien untuk menggambarkan suatu algoritma tanpa menggunakan tatacara penulisan bahasa pemrograman tertentu. Pseudocode tidak dapat dieksekusi langsung pada komputer, tetapi merupakan model yang harus diubah menjadi bahasa pemrograman yang sebenarnya.

Pseudocode ditulis dengan bahasa yang mudah dipahami (boleh menggunakan bahasa alami suatu daerah tertentu), agar alur logika yang digambarkan dapat lebih mudah dimengerti. Pseudocode disusun dengan tujuan untuk menggambarkan tahap-tahap penyelesaian masalah dengan kata-kata (teks). Tetapi metode ini memiliki kelemahan karena penyusunannya sangat dipengaruhi oleh tata bahasa pembuatnya sehingga kadang sulit dipahami oleh orang lain yang ingin mempelajarinya.

Pseudocode secara alamiah dapat terdiri dari berbagai bentuk, walaupun banyak meminjam tatacara penulisan dari bahasa pemrograman popular seperti C, Lisp dan Fortran.

Aturan penulisan teks algoritma

Tidak ada notasi yang baku dalam penulisan teks algoritma. Algoritma bukanlah program yang harus mengikuti aturan-aturan tertentu tetapi dituliskan mendekati gaya bahasa pemrograman secara umum. Teks algortima disusun dalam 3 bagian, yaitu:

a. Bagian Kepala Algoritma

   Kepala algoritma terdiri dari nama algoritma yang berisi penjelasan tentang algoritma yang menguraikan secara singkat hal-hal yang dilakukan oleh algoritma.

b. Bagian Deklarasi

Berisi semua nama yang digunakan, meliputi nama-nama tipe, konstanta, variable dan sub program.

c. Bagian Deskripsi Algortima

Berisi semua langkah-langkah atau aksi untuk menyelesaikan masalah. Algoritma dibaca dari atas ke bawah.

Setiap bagian disertai dengan penjelasan atau dokumentasi tentang maksud pembuatan teks. Bagian penjelasan dituliskan di dalam kurung kurawal {}.

Contoh: algoritma untuk menghitung luas lingkaran

```
Algoritma luas_lingkaran {kepala program}
  {bagian deklarasi}
    Const
      Phi ← 3.14
    Var
      R, L: Real;
  {bagian deskripsi}
    Begin
      {input data R}
      Read(R)
      {proses}
      If R<= 0 then
        write("Data Salah")
      else
                    L ← phi * R * R
      End if
  {output}
      Write (L)
End.
```

1.3. Bahasa Pemrograman

Program adalah kumpulan instruksi/ perintah yang disusun sebagai satu kesatuan prosedur yang berupa urutan langkah-langkah untuk menyelesaikan suatu masalah sehingga dapat di eksekusi oleh komputer. Program memberitahukan kepada komputer apa yang harus dilakukan.

Pemrograman adalah proses mengimplementasikan urutan langkah untuk menyelesaikan suatu masalah dengan menggunakan bahasa pemrograman.

Bahasa pemrograman adalah prosedur/ tatacara penulisan program komputer. Bahasa pemrograman berfungsi sebagai media untuk mneyusun dan memahami suatu program komputer serta sebagai alat komunikasi antara programmer dengan komputer.

Pemakaian bahasa pemrograman selalu berhubungan dengan:

a. Sintak,
   urutan gramatikal yang mengatur tatacara penulisan kata, ekspresi dan pernyataan.
b. Semantik
   Aturan-aturan untuk menyatakan suatu arti yang terkandung dalam sebuah statement/ pernyataan.
c. Kebenaran Logika
   Apakah urutan langkah-langkah dan logika sudah benar dan sesuai dengan penyelesaian masalahnya.

Bahasa pemrograman digolongkan menjadi:

a. Bahasa Mesin
   Bahasa yang dapat dimengerti oleh komputer berupa bahasa numerik. Setiap jenis komputer memiliki bahasa mesin yang berbeda.
b. Bahasa Tingkat Rendah /BTR (Low Level Language)

Bahasa pemrograman yang berorientasi pada mesin. Bahasa ini lebih dekat ke bahasa mesin daripada bahasa manusia. Contohnya bahasa assembly.

Kelebihan BTR:

- Kecepatan eksekusi sangat tinggi
- Executable file yang dihasilkan ukurannya paling kecil

Kekurangan BTR:

- Sulit dipelajari
- Bahasa assembly untuk setiap jenis mikroprosesor satu dengan yang lain sangat jauh berbeda karena belum ada standarisasi.
- Fungsi-fungsi yang tersedia sangat terbatas

c. Bahasa Tingkat Tinggi /BTT (High Level Language)

Bahasa pemrograman yang lebih dekat dengan bahasa manusia daripada bahasa mesin. BTT merupakan bahasa pemrograman yang memiliki aturan-aturan gramatikal dalam penulisan ekspresi atau pernyataan dengan standart yang mudah dipahami oleh manusia. Contoh: Basic, Fortran, Cobol, Pascal, Prolog, dll.

Kelebihan BTT:

- Mudah dipelajari
- Mempunyai fasilitas trace and debug untuk mendeteksi adanya kesalahan
- Mempunyai fungsi/ library yang lengkap sehingga dapat mempermudah dan mempercepat pembuatan program

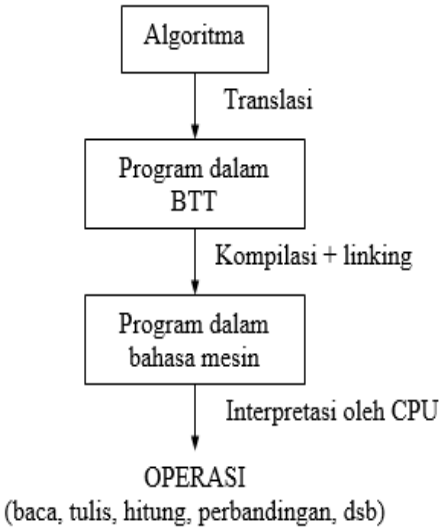
d. Bahasa Tingkat Menengah (Middle Level Language)

Bahasa pemrograman yang mempunyai kelebihan hampir menyamai bahasa assembly karena kelengkapan fungsinya dalam mengakses perangkat keras. Contohnya bahasa C, C++.

Eksekusi program oleh komputer, dilakukan dengan cara menterjemahkan bahasa pemrograman BTT menjadi kode bahasa

mesin. Yang bertugas menterjemahkan disebut Interpreter dan Compiler. Kedua penterjemah ini membaca perintah source code baris per baris kemudian menterjemahkannya menjadi kode mesin yang bersesuaian. Tetapi bedanya pada Interpreter kode mesin dijalankan seketika dan tidak disimpan, sedangkan pada Compiler kode mesin disimpan.

Eksekusi program oleh kompuetr digambarkan sebagai berikut:



Gambar 1. 1. Eksekusi Program

Macam-macam teknik pemrograman:

1. Pemrograan Terstruktur, merupakan teknik yang terstruktur dan sistematis, merupakan proses mengimplementasikan urutan langkah untuk menyelesaikan suatu masalah dalam bentuk program yang memiliki rancang bangun yang terstruktur dan tidak berbelit-belit sehingga mudah ditelusuri dan dipahami serta dikembangkan oleh siapa saja.

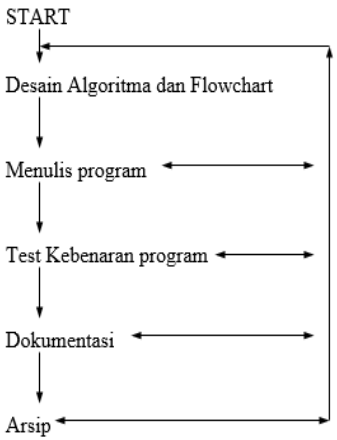
   Ciri-ciri dari pemrograman terstruktur adalah:

   a. Mengandung algoritma pemecahan masalah yang tepat dan benar, *standart* dan efektif.

b. Memiliki struktur logika dan struktur program yang benar dan mudah dipahami.
c. Memiliki dokumentasi yang baik.

2. Pemrograman *Modular*, pemrograman dengan pembagian masalah yang besar dan komplek menjadi kelompok masalah yang lebih kecil yang disebut modul. Teknik pemrograman terstruktur yang digunakan untuk menyelesaikan masalah tersebut dinamakan teknik pemrograman modular, tetapi setelah masing-masing modul disusun maka harus dibuat suatu sistem untuk mengintegrasikannya sehingga menjadi satu kesatuan program yang lengkap. Modul program adalah sekumpulan instruksi yang memiliki operasi-operasi dan data yang didefinisikan memiliki struktur internal yang tidak tergantung pada sub program lain dan merupakan satu kesatuan yang akan dieksekusi secara berulang-ulang. Untuk menyusun program *modular* dapat digunakan konsep fungsi atau prosedur.
3. Pemrograman Berorientasi Object, pemrograman berorientasi object diciptakan supaya konsep yang ada pada dunia nyata dapat diterapkan dalam pemrograman. Dalam pemrograman berorientasi object komponen-komponen dalam program disebut sebagai sebuah object yaitu sesuatu yang memiliki sifat (*property*), kerja (*methode*), dan respon terhadap kejadian (*event*), Object tersebut disusun dari sekumpulan data, prosedur dan fungsi yang dibungkus menjadi satu.

Langkah-langkah dalam proses pembuatan suatu program dijelaskan pada gambar dibawah ini:



Gambar 1. 2. Langkah Membuat Program

1.4. Tipe Data dan Operator

Sebuah program selalu berhubungan dengan data untuk diinputkan, diolah dan kemudian memberikan hasil yang diinginkan. Variabel dan konstanta merupakan penampung data di dalam program. Variabel bersifat dinamis sedangkan konstanta bersifat statis.

Setiap data yang disimpan di dalm variabel atau konstanta harus memiliki tipe data. Ada beberapa tipe data sederhana yang dapat digunakan dalam Pascal. Berikut adalah tipe data yang sering digunakan.

a. Tipe bilangan bulat dapat dikelompokan menjadi:

   Tipe: Ukuran memory: kawasan (range):

   - Byte 1 byte 0…255
   - Word 2 byte 0…65535
   - ShortInt 1 byte -128…127
   - Integer 2 byte -32768…32767
   - LongInt 4 byte -2147483638…2147483647

b. Tipe bilangan pecahan (real)

c. Tipe string yaitu data yang berisi nol atau beberapa karakter. Tipe string diapit dengan tanda petik(‘….’).contoh:’nama’.

d. Tipe Char yaitu untuk data yang berisi hanya sebuah karakter saja. Misalnya:'A','1','?'.
e. Tipe Boolean adalah suatu data yang nilainya berupa false (salah) atau true (benar).

Disamping tipe –tipe tersebut masih terdapat beberapa tipe lainnya dan dapat juga tipe tersebut dibuat sendiri.

Data-data dalam program akan diproses dengan menggunakan operator. Operator adalah simbol atau kata yang digunakan dalam program untuk melakukan suatu operasi data seperti penjumlahan, pengurangan, pemberian nilai ke dalam variabel, membandingkan kesamaan dua buah nilai dan sebagainya.

Nilai data yang dioperasikan oleh operator bersama operand membentuk suatu ekspresi. Operator yang digunakan dalam Pascal:

Tabel 1. 1. Operator Matematik

Operator Matematik		
Simbol	Keterangan	Prioritas
*	Perkalian	1
/	Pembagian bilangan real	1
DIV	Pembagian bilangan bulat	1
Mod	Sisa hasil pembagian	1
+	Penjumlahan	2
-	Pengurangan	2

Tabel 1. 2. Perbandingan

Operator Perbandingan	
>	Lebih besar
<	Lebih kecil
>=	Lebih besar sama dengan
<=	Lebih kecil sama dengan
<>	Tidak sama dengan
=	Sama dengan

Tabel 1. 3. Operator Logika

Operator Logika	
AND	Dan
OR	Atau
NOT	Bukan

1.5. Soal Latihan

1. Jelaskan apa yang dimaksud dengan logika?
2. Jelaskan apa yang dimaksud dengan Pseudocode!
3. Jelaskan apa yang dimaksud dengan Bahasa pemrograman!
4. Sebutkan tipe data dan operator yang ada pada pemrograman!

BAB II

STRUKTUR DATA

Capaian Pembelajaran:

1. Mampu menjelaskan istilah-istilah dasar, dasar pengorganisasian data, struktur data, operasi struktur data dan algoritma.
2. Mampu membuat struktur runtutan
3. Mampu membuat teknik runtutan

Data adalah suatu nilai atau kumpulan nilai. Suatu item data merujuk pada satu tipe nilai. Item data yang terbagi atas beberapa subitem disebut Group item. Sebagai contoh Alamat bisa dibagi menjadi 3 bagian yaitu nama jalan, kota dan propinsi.

Kumpulan data biasanya disusun ke dalam bentuk hirarki dari field, records dan files. Dalam struktur data hirarki disusun ke dalam dalam bentuk atribut, entitas dan kumpulan entitas.

Entitas adalah sesuatu yang mempunyai atribut atau properti tertentu yang mungkin mengandung nilai. Nilai itu sendiri bisa berupa nilai numerik atau non numerik. Sebagai contoh berikut ini adalah atribut dan nilai dari seorang karyawan:

```
Atribut:   Nama  Umur  Jenis Kelamin    No. KTP
Nilai:     JOHN BROWN 34    Pria        123-45-
6789
```

Entitas dengan atribut yang sama (dalam hal ini seluruh karyawan) membentuk suatu kumpulan entitas. Setiap atribut dari suatu kumpulan entitas mempunyai range nilai tertentu.

Cara data disusun dalam hirarki field, record dan file menggambarkan relasi yang sama antara atribut, entitas dan kumpulan

entitas. Suatu field berasosiasi dengan atribut dari suatu entitas, record berasosisai dengan suatu entitas, sedangkan file berasosiasi dengan kumpulan entitas.

Suatu record dalam suatu file dapat mengandung beberapa field, field tertentu dapat berlaku unik yang membedakan suatu record dengan record lainnya. Field tersebut disebut dengan primary key. Contoh kasus sebagai berikut:

a. Suatu dealer mobil mengelola data mobil-mobil, dimana setiap record mempunyai field sebagai berikut:

   No. Rangka, Type, Tahun, Harga, Aksesoris

   Field Nomor rangka dapat berlaku sebagai *primary key* karena setiap mobil mempunyai nomor rangka yang berbeda.

b. Suatu perusahaan mengelola data karyawannya, dimana setiap record mempunyai field sebagai berikut:

   Nama, Alamat, No. Telepon

   Walaupun terdapat 3 data item, Nama dan alamat sebenarnya dapat membentuk group item yang dapat berlaku sebagai *primary key*. Sedangkan Alamat dan No. Telepon tidak dapat dijadikan *primary key* karena mungkin saja karyawan mempunyai alamat dan no. Telepon yang sama (satu keluarga).

Record dapat diklasifikasikan berdasarkan panjangnya. Suatu file dapat mempunyai record dengan panjang yang tetap (*fixed-length records*) atau record dengan panjang tidak tetap (*variable-length records*). Dalam *fixed-length record*, semua record mempunyai item data yang sama dengan ukuran data yang sama untuk tiap item datanya. Sedangkan *variable-length records*, record mempunyai panjang yang yang berbeda. Sebagai contoh, data mahasiswa dikategorikan dalam *variable-length record* karena setiap siswa mengambil sejumlah matakuliah yang berbeda. Biasanya variable-length records sudah ditentukan panjang minimum dan maksimumnya.

Organisasi data dalam bentuk fields, record dan file hanyalah salah bentuk pengelolaan data. Data juga diorganisasikan dalam bentuk yang lebih kompleks. Bagian berikut dari bab ini akan menjelaskan secara singkat berbagai bentuk struktur data tersebut.

2.1. Struktur Data

Struktur Data adalah model logika atau matematika dari suatu organisasi data tertentu. Pemilihan bentuk model data tertentu tergantung pada dua pertimbangan. Pertama, model ini harus dapat mencerminkan hubungan antara data dan dunia nyata. Selain itu struktur tersebut harus cukup sederhana sehingga dapat diproses secara efektif ketika diperlukan. Berikut ini akan dijelaskan secara singkat mengenai beberapa struktur data.

Arrays

Array merupakan struktur data yang paling sederhana. Data disusun berdasarkan n jumlah data terbatas, dimana setiap elemen data ditunjukkan oleh urutan angka, biasanya 1, 2, 3.. n. Jika kita mendeklarasikan suatu array dengan A, maka elemen-elemen dari A dinotasikan dalam notasi kurung siku

```
A[1], A[2], A[3],... A[N]
```

Berdasarkan notasi tersebut, angka N dari A[N] disebut subscript dan A[N] disebut *subscripted variables*.

Contoh:

Suatu array dengan nama MAHASISWA mengandung 6 nama mahasiswa seperti yang terlihat pada Error! Reference source not found. .

Gambar 2. 1. Contoh Array

Array diatas disebut array satu dimensi (*one dimensional array*) karena setiap elemen dalam array ditunjukkan oleh satu *subscript*. Array 2 dimensi adalah kumpulan elemen data yang sama dimana setiap elemen ditunjukkan oleh 2 *subscript*. Dalam matematika, array tersebut dikenal dengan matriks atau tabel dalam istilah bisnis terapan.

Contoh:

Makro mempunyai 17 Cabang dimana setiap cabang mempunyai 4 departemen. Penjualan mingguan Makro untuk masing-masing cabang dan departemen dapat digambarkan menggunakan array 2 dimensi (lihat **Error! Reference source not found.**) dimana subscript pertama menunjukkan cabang dan subscript kedua menunjukkan departemen. Jika array tersebut dideklarasikan dengan nama SALES maka:

SALES[1,1]:= 2872, SALES[1,2]:= 85,..., SALES[28,4]:= 982

Array tersebut mempunyai 112 elemen karena terdiri dari 28 baris dan 4 kolom.

Dept. / Cabang	1	2	3	4
1	2872	805	3211	1560
2	2196	1223	2525	1744
3	3257	1017	3686	1951
...	...	...	...	...
28	2618	931	2333	982

Gambar 2. 2. Contoh Array

Linked List

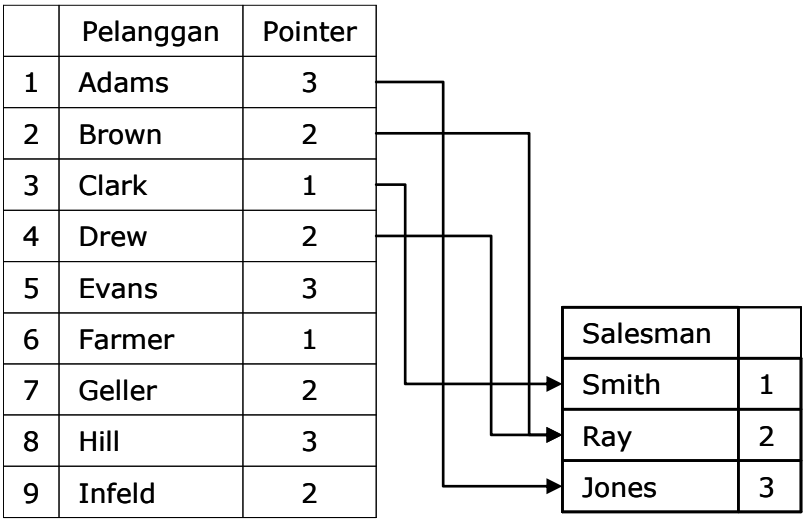
Sebuah perusahaan distribusi mengelola data dimana setiap record mengandung nama pelanggan dan salesman yang menanganinya seperti terlihat pada gambar berikut:

	Pelanggan	Salesman
1	Adams	Smith
2	Brown	Ray
3	Clark	Jones
4	Drew	Ray
5	Evans	Smith
6	Farmer	Jones
7	Geller	Ray
8	Hill	Smith
9	Infeld	Ray

Gambar 2. 3. Linked List

Struktur tersebut dapat saja disimpan ke dalam suatu file komputer, akan tetapi itu bukan merupakan cara terbaik karena nama salesman diulang beberapa kali dalam beberapa records. Cara lain untuk merepresentasikan data tersebut adalah dengan jalan memisahkan salesman dan menambahkan field Pointer pada file pelanggan yang menunjukkan lokasi masing-masing salesman, lihat **Error! Reference source not found.**. Penggunaan pointer yang bertype data integer,

memerlukan lebih sedikit ruang bila dibandingkan nama salesman, terutama jika terdapat ratusan pelanggan untuk tiap salesman.



	Pelanggan	Pointer
1	Adams	3
2	Brown	2
3	Clark	1
4	Drew	2
5	Evans	3
6	Farmer	1
7	Geller	2
8	Hill	3
9	Infeld	2

Salesman	
Smith	1
Ray	2
Jones	3

Gambar 2. 4. Struktur Linked List

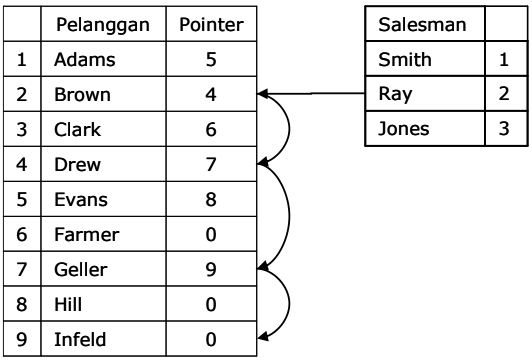
Misalkan perusahaan ingin melihat daftar semua pelanggan berdasarkan masing-masing salesman. Jika menggunakan bentuk di atas, kita harus mencari ke seluruh record dalam file pelanggan. Ada satu cara untuk menyederhanakan pencarian yaitu dengan membalik arah panah, tiap salesman akan memiliki beberapa pointer sesuai dengan pelanggannya. Kerugian penggunaan struktur data ini adalah tiap salesman mempunyai banyak pointer dan susunan pointer akan berubah apabila ada penambahan atau pengurangan pelanggan.

	Salesman	Pointer
1	Smith	3, 6
2	Ray	2, 4, 7, 9
3	Jones	1, 5, 8

Gambar 2. 5. Daftar Semua Pelanggan

Cara lain yang lebih populer untuk menyimpan data di atas adalah dimana setiap salesman mempunyai satu pointer yang menunjuk

pelanggan pertamanya, setiap pelanggan yang ditunjuk mempunyai satu link yang menunjuk ke pelanggan berikutnya dan seterusnya, pelanggan terakhir ditandai dengan link 0. Dengan menggunakan cara ini kita dapat dengan mudah memperoleh daftar seluruh pelanggan dari seorang salesman selain itu pelanggan dapat dengan mudah disisipkan atau dihapus dari daftar.



	Pelanggan	Pointer
1	Adams	5
2	Brown	4
3	Clark	6
4	Drew	7
5	Evans	8
6	Farmer	0
7	Geller	9
8	Hill	0
9	Infeld	0

Salesman	
Smith	1
Ray	2
Jones	3

Gambar 2. 6. Contoh Linked List

Representasi data pada **Error! Reference source not found.** merupakan salah satu contoh link list. Istilah pointer dan link kadang digunakan secara bersamaan. Istilah pointer digunakan untuk menunjuk suatu elemen pada tabel lain, sedangkan link digunakan untuk menunjuk elemen yang terdapat pada tabel yang sama.

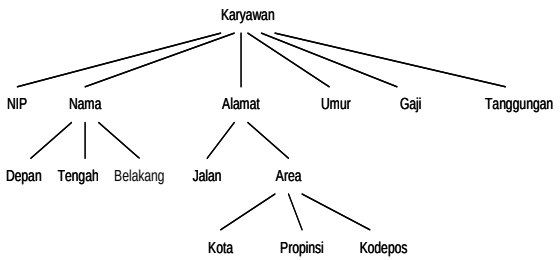
Tree

Data kadang mengandung hubungan hirarki antar elemen yang berbeda. Struktur data yang menggambarkan hubungan ini disebut tree.

Sebagai contoh, seorang karyawan mempunyai record dengan item data sebagai berikut:

```
NIP, Nama, Alamat, Umur, Gaji, Tanggungan
```

Nama adalah suatu group item yang terdiri dari nama depan, nama tengah dan nama belakang, demikian juga dengan alamat merupakan group item yang terdiri dari subitem Nama Jalan dan Area, dimana Area juga merupakan group item yang terdiri dari Kota, Propinsi dan Kodepos. Data tersebut dapat digambarkan menggunakan struktur data tree sebagai berikut:



Gambar 2. 7. Contoh Struktur Tree

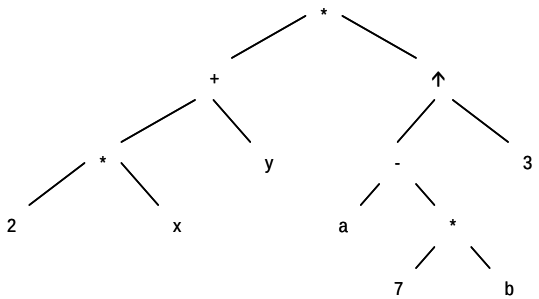
Atau

```
01 Karyawan
    02 NIP
    02 Nama
        03 Depan
        03 Tengah
        03 Belakang
    02 Alamat
        03 Jalan
        03 Area
            04 Kota
            04 Propinsi
            04 Kodepos
    02 Umur
    02 Gaji
    02 Tanggungan
```

Contoh lain penggunaan tree adalah ekspresi matematika. Misalkan terdapat ekspresi matematika sebagai berikut:

$(2x + y)(a - 7b)^3$

Ekspresi tersebut dapat digambarkan menggunakan tree sebagai berikut:



Gambar 2. 8. Contoh Struktur Tree

Stack

Stack atau tumpukan adalah suatu linear list dimana penyisipan dan penghapusan elemen hanya dapat dilakukan pada bagian atas list (*top*). Struktur ini mirip dengan tumpukan piring dimana penambahan dan pengambilan piring dilakukan pada bagian atas (menggunakan sistem *LIFO – Last in First Out*).

Queue

Queue atau antrian adalah suatu linear list dimana penyisipan dilakukan pada bagian belakang list (*rear*) dan penghapusan elemen dilakukan pada bagian depan list (*front*). Struktur ini mirip dengan antrian orang membeli karcis pertunjukan, yang pertama antri akan dilayani terlebih dahulu (menggunakan sistem *FIFO – First In First Out*).

Graph

Data kadang-kadang mengandung hubungan berpasangan antar elemen yang tidak dapat digambarkan secara hirarki. Sebagai contoh,

arah penerbangan pesawat yang digambarkan dengan menarik garis antar kota. Struktur data seperti ini dinamakan Graph.

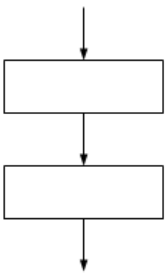
2.2. Struktur Runtutan

Pada struktur runtutan/ berurutan setiap pernyataan akan dikerjakan satu per satu sesuai dengan urutan penulisan algoritma program. Setiap pernyataan dilaksanakan tepat satu kali dan tidak ada pernyataan yang diulang atau pernyataan yang tidak dilaksanakan. Akhir dari pernyataan terakhir merupakan akhir dari algoritma program tersebut.

Jika pernyataan dilambangkan dengan A_1, A_2, A_3……., A_n, maka pelaksanaan instruksi dapat digambarkan sebagai berikut:

Gambar 2. 9. Contoh Struktur Runtutan

Jika digambarkan dalam flowchart:



Gambar 2. 10. Flowchart Struktur Runtutan

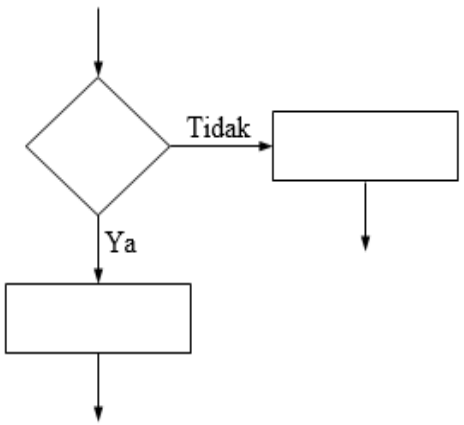
Struktur Percabangan

Pada struktur percabangan memungkinkan suatu pernyataan untuk dieksekusi hanya jika suatu kondisi tertentu terpenuhi atau tidak

terpenuhi. Struktur percabangan disebut juga dengan struktur pemilihan. Terdapat beberapa pernyataan yang akan dipilih berdasarkan suatu kondisi. Sehingga terdapat bagian pernyataan yang dilewati atau tidak dijalankan oleh program.

Terdapat dua macam instruksi struktur percabangan dalam pemrograman, yaitu IF dan Case yang akan kita pelajari pada bab selanjutnya.

Jika digambarkan dalam flowchart:



Gambar 2. 11. Struktur Percabangan

Struktur Perulangan

Struktur perulangan merupakan struktur algoritma yang akan melakukan suatu proses/ eksekusi yang berulang-ulang jika suatu kondisi terpenuhi atau tidak terpenuhi. Proses perulangan ini biasanya digunakan untuk:

a. Mengulang proses pemasukan data
b. Mengulang proses perhitungan
c. Mengulang Proses penampilan hasil pengolahan data

Struktur perulangan terdiri dari:

a. Inisialisasi

   Aksi yang dilakukan sebelum pengulangan dilakukan pertama kali.

b. Kondisi pengulangan

   Ekspresi boolean yang harus dipenuhi untuk melaksanakan pengulangan.

c. Badan pengulangan

   Satu atau lebih aksi/ pernyataan yang akan diulang.

d. Terminasi

Aksi yang dilakukan setelah pengulangan selesai dilakukan.

Bentuk umum perulangan:

<Inisialisasi>

Awal perulangan

Badan perulangan

Akhir perulangan

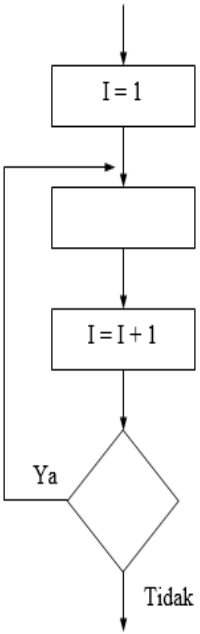
<Terminasi>

Kondisi perulangan dapat berada pada awal perulangan atau akhir perulangan.

Bentuk instruksi pengulangan dan pemrograman terstruktur ada tiga macam, yaitu:

a. For – Next
b. While – Do
c. Repeat – Until

Jika digambarkan dalam flowchart:



Gambar 2. 12. Struktur Perulangan

2.3. Teknik Runtutuan

Bentuk umum penulisan struktur runtutan:

Pernyataan 1;
Pernyataan 2;
Pernyataan 3;
.
.
.
Pernyataan n;

Contoh: Menghitung luas persegi panjang

Panjang:=10;
Lebar:=5;
*Luas:=panjang*lebar;*
Writeln ('Luas=', Luas);

Membaca/ menulis dari/ ke arsip

a. Membuka arsip untuk dibaca

```
Var fin:text;
Begin
        Assign(fin,'data.txt');
        Reset(fin)
        ……
        ……
End.
```

b. Membuka arsip unutk ditulis

```
Var fout:text;
Begin
        Assign(fout,'hasil.txt');
        Rewrite(fout)
        ……
        ……
End.
```

c. Menutup arsip

```
Close (fin);
Close (fout);
```

d. Pernyataan untuk membaca data dari arsip

```
Read (fin,x);
```

e. Pernyataan untuk menulis data ke dalam arsip

```
Write (fout,x);
```

Contoh:

```
Program luas_persegi_panjang;
Var
        Panjang, Lebar: real;
```

```
        Luas: real;
        Fin, Fout: text;
Begin
        { membuka arsip masukan}
                Assign (Fin, 'data.txt');
                Reset (Fin);
        {membuka arsip keluaran}
                Assign (Fout, 'hasil.txt');
                Rewrite (Fout);
        {membaca panjang dan lebar dari arsip Fin}
                Read (Fin, panjang, lebar);
        {menghitung luas}
                Luas:= panjang * lebar;
        {menulis hasil perhitungan luas ke dalam
arsip Fout}
                Writeln (Fout,'Luas persegi panjang
        = ', Luas);
        {menutup arsip}
                Close (Fin);
                Close (Fout);
                End.
```

2.4. Asas-Asas Dalam Kontrak Perjanjian

Asas dalam hukum perjanjian/kontrak dapat dikatakan sebagai batu uji bagi norma hukum yang ada, dalam arti norma hukum tersebut pada akhirnya harus dapat dikembalikan pada asas hukum yang menjiwainya.

Apabila dilihat dari segi fungsinya, maka asas dalam hukum perjanjian/kontrak memiliki 2 (dua) fungsi, yaitu Pertama, membangun fondasi bagi kontruksi hukum kontrak yang kokoh, yang menempatkan kedudukan hukum para pihak yang membuat kontrak dalam hubungan-hubungan hukum kontekstual yang setara, jelas dan konkrit. Kedua,

mengarahkan para pihak yang membuat perjanjian/kontrak untuk mementukan substansi (isi)-nya yang memuat hak dan kewajiban serta hubungan hukum yang tidak bertentangan dengan undang-undang, ketertiban umum dan kesusilaan.

Ada beberapa Asas Dalam Kontrak Perjanjian:

1. Asas kebebasan berkontrak

Asas kebebasan berkontrak ini dikenal dengan istilah "partij otonomie" atau "freedom of contract" atau "liberty of contract". Pada dasarnya asas ini bersifat universal dikarenakan digunakan disemua negara pada umumnya.

Adapun latar belakang lahirnya asas kebebasan berkontrak adalah adanya paham individualisme yang lahir pada zaman Yunani, yang diteruskan oleh kaum Epicuristen dan berkembang pesat pada zaman Renaisance melalui ajaran-ajaran Hugo de Groth, Thomas Hobbes, Jhon Locke dan Rosseau. Menurut paham Individualisme, sistem orang bebas untuk memperoleh apa yang dikehendakinya. Dalam hukum kontrak asas ini diwujudkan dalam "kebebasan berkontrak". Asas kebebasan berkontrak ini bermakna bahwa setiap orang bebas membuat perjanjian dengan siapapun, apapun isinya, apapun bentuknya sepanjang tidak melanggar undang-undang, ketertiban umum dan kesusilaan.

Salah satu dasar hukum untuk melihat diberlakukannya asas kebebasan berkontrak tersebut adalah Pasal 1338 ayat (1) KUHPerdata yang menyatakan bahwa: "semua perjanjian yang dibuat secara sah berlaku sebagai undang-undang bagi mereka yang membuatnya." Kebebasan berkontrak yang diatur dalam Pasal 1338 ayat (1) tersebut pada dasarnya:

a. Memberikan kebebasan untuk membuat atau tidak membuat perjanjian;
b. Mengadakan perjanjian dengan siapapun;
c. Menentukan isi perjanjian, pelaksanaan, dan persyaratannya,
d. Menentukan bentuk perjanjian, yaitu tertulis atau lisan.

e. Keempat hal tersebut dapat dilakukan dengan syarat tidak melanggar undang-undang, ketertiban umum, dan kesusilaan.

2. Asas Konsensualisme

Asas konsensualisme ini berasal dari kata latin "concensus" yang artinya sepakat. Dalam mebuat kontrak disyaratkan adanya consensus, yaitu para pihak sepakat atau setuju mengenai prestasi yang dijanjikan atau dapat diartikan, perjanjian/kontrak tersebut di dadasari adanya kata "sepakat dari kedua pihak". Asas konsensualisme didasarkan Pasal 1320 ayat (1) KUH Perdata yang menyebutkan salah satu syarat dalam perjanjian adalah adanya "kesepakatan kedua pihak".

Walaupun terjadi kesepakatan, tetapi perlu tetap diperhatikan unsur "kehendak" dalam melakukan kesepakatan tersebut. Apabila kehendak melakukan perjanjian/kontrak atas dasar kedua belah pihak, maka perjanjian dianggap sah. Namun, apabila perjanjian/kontrak yang dilakukan dengan adanya paksaan (conradictio interminis), maka perjanjian/kontrak tersebut dapat dibatalkan dengan memohon kepada pengadilan. Dalam KUHPerdata terdapat hal-hal yang dapat dikategori dengan "cacat kehendak" yang membuat perjanjian/kontrak dapat dibatalkan, yaitu:

a. Kesesatan (dwaling);
b. Penipuan atau (bedrog); serta
c. Paksaan atau (dwang).

3. Asas kekuatan hukum mengikat perjanjian/kontrak

Asas kekuatan mengikat perjanjian/kontrak mengharuskan para pihak memenuhi apa yang telah merupakan ikatan mereka satu sama lain dalam kontrak yang mereka buat. Asas hukum ini disebut juga asas pacta sunt servanda, yang secara konkrit dapat dicermati dalam Pasal 1338 ayat (1) KUHPerdata yang memuat ketentuan imperatif, "semua kontrak yang dibuat sesuai dengan undang-undang berlaku sebagai undang-undang bagi mereka yang membuatnya."

Adapun pacta sunt servanda diakui sebagai aturan yang yang terkandung didalamnya, dimaksudkan untuk dilaksanakan dan pada akhirnya dapat dipaksakan penataannya. Asas ini menimbulkan kepastian hukum bagi para pihak yang telah memperjanjikan sesuatu memperoleh kepastian bahwa perjanjian itu dijamin pelaksanaannya. Hal ini sesuai dengan kekuatan Pasal 1338 KUH Perdata, yang intinya menyebutkan bahwa perjanjian tidak dapat ditarik kembali selain diperbolehkan oleh undang-undang.

Asas ini dapat berlaku apabila kedudukan para pihak tidak seimbang. Tetapi jika kedudukan para pihak seimbang maka undang-undang memberi perlindungan bahwa perjanjian itu dapat dibatalkan, baik atas tuntutan para pihak yang dirugikan, kecuali dapat dibuktikan pihak yang dirugikan menyadari sepenuhnya akibat-akibat yang timbul.

4. Asas Itikat Baik

Asas itikad baik (in good faith) merupakan salah satu asas yang dikenal dalam hukum perjanjian/kontrak. Ketentuan ini diatur dalam pasal 1338 ayat (3) KUH Perdata, bahwa perjanjian harus dilaksanakan dengan itikad baik.

Dalam pelaksanaan perjanjian, asas itikad baik mempunyai dua pengertian yaitu:

a. Itikad baik dalam pengertian subyektif. Merupakan sikap batin seseorang pada saat dimulainya suatu hubungan hukum berupa perkiraan bahwa syarat-syarat yang telah diperlukan telah dipenuhi, di sini berarti adanya sikap jujur dan tidak bermaksud menyembunyikan sesuatu yang buruk yang dapat merugikan pihak lain.
b. Itikad baik dalam pengertian obyektif. Ini merupakan tindakan seseorang dalam melaksanakan perjanjian yaitu pada saat melaksanakan hak dan kewajiban dalam suatu hubungan hukum.Artinya bahwa pelaksanaan perjanjian harus berjalan di atas ketentuan yang benar, yaitu mengindahkan norma-norma kepatutan dan kesusilaan.Asas itikad baik ini diatur dalam Pasal 1338 ayat (3)

KUHPerdata yang menentukan bahwa persetujuan harus dilakukan dengan itikad baik.Dari ketentuan di atas, hakim diberi wewenang untuk mengawasi pelaksanaan perjanjian, jangan sampai pelaksanaan itu melanggar kepatutan dan keadilan.

2.5. Operasi Struktur Data

Operasi Struktur data yang utama adalah sebagai berikut:

a. *Traversing*: Menjelajahi tiap record sehingga record tersebut dapat diproses.
b. *Searching*: Mencari lokasi record berdasarkan nilai kunci, atau mencari lokasi seluruh record yang memenuhi satu atau lebih kondisi.
c. *Inserting*: Menyisipkan sebuah record ke dalam struktur.
d. *Deleting*: Menghapus sebuah record dari struktur.

Kadang-kadang dua atau lebih operasi digunakan untuk situasi tertentu. Misalkan kita ingin menghapus record berdasarkan nilai kunci tertentu maka pertama kita harus melakukan searching untuk mencari lokasi record tersebut setelah ditemukan baru kita melakukan penghapusan.

Selain empat operasi di atas, operasi berikut ini digunakan untuk keadaan khusus yaitu:

a. *Sorting*: Mengurutkan record dalam urutan logika tertentu.
b. *Merging*: Menggabungkan record dari dua file terurut yang berbeda menjadi satu file yang terurut.

Contoh:

Suatu organisasi mengelola data keanggotaannya, dimana setiap record mengandung data sebagai berikut:

Nama, Alamat, No. Telepon, Umur dan Jenis Kelamin.

a. Apabila pimpinan ingin mengumumkan pertemuan melalui surat. Ia harus menjelajahi record untuk mendapatkan nama dan alamat anggota.
b. Apabila seseorang ingin mencari nama-nama anggota yang tinggal di area tertentu, maka ia harus menjelajahi tiap record untuk mendapatkan data tersebut.
c. Apabila seseorang ingin mengetahui alamat dari seorang anggota, maka ia harus melakukan searching.
d. Apabila ada seseorang bergabung dengan organisasi, maka seseorang harus menyisipkan record baru ke dalam file.
e. Apabila ada seorang member yang meninggal dunia, maka seseorang harus menghapus record tersebut dari file.
f. Apabila seorang member pindah dan memiliki alamat baru maka seseorang harus melakukan searching record member tersebut kemudian melakukan update yaitu merubah item data sesuai dengan keadaan terbaru.

Apabila seseorang ingin mengetahui jumlah anggota yang berumur di atas 65 tahun maka ia harus menjelajahi file dan menghitung anggota yang diinginkan.

2.6. Algoritma, Kompleksitas dan Pertukaran antara Ruang dan Waktu

Algoritma adalah suatu daftar langkah-langkah yang terdefinisi dengan baik untuk memecahkan masalah tertentu. Salah satu tujuan utama kita mempelajari struktur data adalah untuk membuat algoritma yang efisien untuk pengolahan data. Banyaknya ruang dan waktu yang digunakan merupakan dua hal utama untuk mengukur efisiensi algoritma. Kompleksitas algoritma adalah fungsi yang memberikan running time dan ruang yang berkaitan dengan ukuran input.

Tiap algoritma akan melibatkan struktur data tertentu. Kita tidak dapat selalu menggunakan algoritma yang paling efisien, karena

pemilihan struktur data tergantung dengan banyak hal, termasuk tipe data dan frekwensi penggunaan berbagai operasi data yang akan diterapkan. Kadang-kadang pemilihan struktur data melibatkan pertukaran ruang dan waktu yaitu dengan meningkatkan jumlah ruang untuk menyimpan data dapat mengurangi waktu yang dibutuhkan untuk memproses data atau sebaliknya.

Algoritma Pencarian

Perhatikan

Contoh, dimana tiap record mengandung data nama dan nomor telepon anggotanya. Misalkan kita diminta untuk mencari data nomor telepon berdasarkan nama tertentu. Cara pertama adalah melakukan linear search menggunakan algoritma berikut ini:

Linear Search: Cari di tiap record dari record pertama, sampai ditemukan Nama yang diinginkan.

Pertama, sangat jelas bahwa waktu yang diperlukan untuk menjalankan algoritma adalah sebanding dengan jumlah perbandingan. Selain itu rata-rata perbandingan untuk n records adalah n/2, karena itu kompleksitas algoritma linear search adalah C(n) = n/2.

Algoritma di atas akan mustahil dipraktekkan jika kita mencari dalam daftar yang berisi ribuan data seperti dalam buku telepon. Akan tetapi jika nama diurutkan secara alfabet, seperti dalam buku telepon, maka kita dapat menggunakan algoritma yang lebih efisien yang disebut binary search.

Binary Search: Bandingkan Nama dengan nama pada bagian tengah daftar, ini akan menentukan di bagian mana Nama berada. Kemudian bandingkan Nama dengan nama pada bagian tengah pada setengah

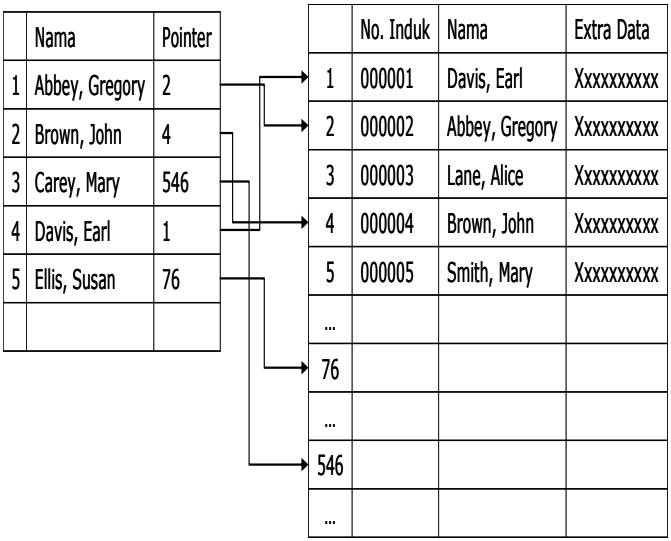
bagian yang benar. Demikian seterusnya sampai ditemukan Nama yang dicari.

Kompleksitas algoritma *binary search* ini adalah C(n) = Log_2 n. Sebagai perbandingan, untuk mencari Nama pada daftar yang berisi 25.000 nama hanya dibutuhkan tidak lebih 15 perbandingan.

Walaupun algoritma *binary search* merupakan algoritma yang sangat efisien, masih ada beberapa hal yang harus dipertimbangkan. Khususnya, algoritma mengasumsikan seseorang harus mencari berdasarkan nama tengah dalam suatu daftar atau sub-daftar. Ini berarti daftar tersebut harus dimasukkan dalam suatu array dengan type data yang sama. Sayangnya, menyisipkan elemen ke dalam suatu array menggeser elemen lainnya ke bawah, sedangkan pada penghapusan elemen dalam suatu array akan menggeser elemen lainnya ke atas.

Pertukaran Ruang dan Waktu

Misalkan suatu file berisi record yang mengandung nama, nomor induk pegawai dan informasi lainnya yang berkaitan. Dengan cara menyortir berdasarkan alfabet dan menggunakan *binary search* merupakan cara terbaik untuk mencari suatu record berdasarkan nama. Di lain sisi, seandainya kita hanya diberi nomor induk pegawai, mau tidak mau kita harus mencari menggunakan *linear search* yang sangat menguras waktu untuk ukuran data yang besar. Bagaimana kita harus memecahkan masalah ini? Salah satu caranya adalah dengan membuat file lain yang disortir berdasarkan nomor induk pegawai, ini berakibat kita harus menyediakan ruang dua kali lebih banyak. Cara lain adalah dengan cara menyediakan file utama yang disortir berdasarkan nomor induk pegawai dan array bantu yang hanya mempunyai dua kolom yaitu kolom nama yang diurutkan berdasarkan alfabet dan kolom yang berisi pointer yang menunjuk ke lokasu yang berkaitan pada file utama. Ini merupakan salah satu cara yang paling banyak digunakan, karena ruang tambahan yang diperlukan hanya terdiri dari dua kolom.



Gambar 2. 13. Pertukaran Ruang dan Waktu

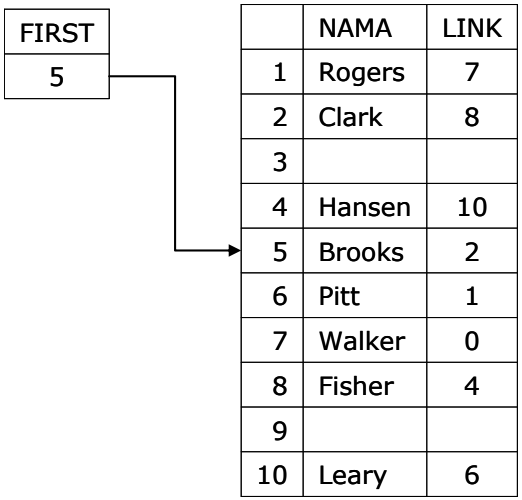
2.7. Soal Latihan

1. Seorang dosen mempunyai daftar mahasiswa, dimana terdiri dari: Nama, Jurusan, NIM, Nilai Test, Nilai Akhir
   a. Tentukan yang mana yang dimaksud dengan entitas, atribut dan kumpulan entitas dari data di atas.
   b. Jelaskan apa yang dimaksud dengan fields, record dan file.
   c. Atribut mana yang dapat berlaku sebagai primary key?
2. Berikan deskripsi singkat untuk:
   a. Traversing
   b. Sorting
   c. Searching
3. Misalkan terdapat array NAMA yang disusun berdasarkan alfabet (lihat gambar).
   a. Cari Nama[2], Nama[4], dan Nama[7].
   b. Misalkan Davis disisipkan ke dalam array. Berapa banyak nama yang harus dipindah ke lokasi baru.

c. Seandainya Gupta dihapus dari array. Berapa banyak nama yang harus dipindah ke lokasi baru.

4. Sebuah rumah sakit mengelola data pasiennya dalam suatu file dimana setiap recordnya mengandung:
   Nama, Tanggal Masuk, No. KTP, Ruang, No. Tempat Tidur, Dokter
   a. Item data mana yang dapat berlaku sebagai primary key?
   b. Pasangan item mana yang dapat berlaku sebagai primary key?
   c. Item mana yang merupakan group item?

5. Berikan deskripsi singkat untuk istilah:
   a. Inserting.
   b. Deleting
6. Misalkan terdapat suatu array Nama (lihat gambar). Nilai FIRST dan LINK[K] menunjukkan urutan dari nama-nama tersebut. FIRST menunjukkan lokasi nama pertama dari daftar, dan LINK[K] menunjukkan lokasi selanjutnya, 0 digunakan untuk menandai akhir dari daftar. Tentukan urutan nama-nama tersebut.



FIRST
5

	NAMA	LINK
1	Rogers	7
2	Clark	8
3		
4	Hansen	10
5	Brooks	2
6	Pitt	1
7	Walker	0
8	Fisher	4
9		
10	Leary	6

BAB III

DASAR STRUKTUR DATA

Capaian Pembelajaran:

1. Mampu menjelaskan fungsi dan notasi matematika.
2. Mampu menjelaskan notasi algoritma.
3. Mampu menjelaskan Struktur control.
4. Mampu menjelaskan kompleksitas algoritma.
5. Mampu menjelaskan Subalgoritma.
6. Mampu menjelaskan variabel dalam Struktur Data.

3.1. Notasi dan Fungsi Matematika

Bagian berikut ini akan membahas berbagai fungsi matematika beserta notasinya yang seringkali terdapat dalam analisa algoritma dan dalam ilmu komputer secara umum.

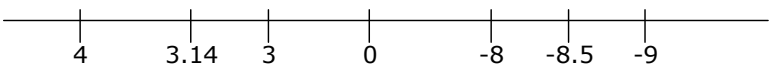
Fungsi Floor dan Ceiling

Jika x adalah sembarang bilangan real, maka x terletak diantara dua integer yang disebut floor dan ceiling dari x.
Secara khusus,

- $\lfloor x \rfloor$ disebut floor dari x yaitu integer terendah yang tidak lebih dari x.
- $\lceil x \rceil$ disebut ceiling dari x yaitu integer tertinggi yang tidak kurang dari x.
- Jika x adalah integer maka $\lfloor x \rfloor = \lceil x \rceil$, jika tidak maka $\lfloor x \rfloor + 1 = \lceil x \rceil$

Contoh:

$\lfloor 3.14 \rfloor = 3$, $\lfloor \sqrt{5} \rfloor = 2$, $\lfloor -8.5 \rfloor = -9$, $\lfloor 7 \rfloor = 7$
$\lceil 3.14 \rceil = 4$, $\lceil \sqrt{5} \rceil = 3$, $\lceil -8.5 \rceil = -8$, $\lceil 7 \rceil = 7$



- Jika x adalah real positif maka ⌊x⌋ akan mendekati 0 dan ⌈x⌉ akan menjauhi 0.
- Jika x adalah real negatif maka ⌊x⌋ akan menjauhi 0 dan ⌈x⌉ akan mendekati 0.

Fungsi Remainder; Aritmatika Modular

Jika k adalah sembarang integer dan M adalah integer positif maka k (mod M) dinotasikan sebagai integer sisa hasil pembagian k dibagi M.

K = Mq + r, dimana $0 \leq r < M$ atau r = k - Mq

Akan tetapi jika k negatif maka -k (mod M) = M – r' dimana r' = k – Mq.

Contoh:

```
25 (mod 7) = 4, 25 (mod 5) = 0, 35 (mod 11) = 2,
3 (mod 8) = 3.
-25 (mod 7) = 7 – 4 = 3, -25 (mod 5) = 5 – 0 =
5, -35 (mod 11) = 11 – 2 = 9
```

M disebut modulus, dan a ≡ b (mod M) dibaca sebagai 'a kongruen b Modulo M'. Persamaan kongruen yang biasa digunakan adalah:

0 ≡ M (mod M) dan a ± M ≡ a (mod M)

Aritmatik Modulo M mengacu pada operasi aritmatik seperti perkalian, penambahan dan pengurangan dimana nilai aritmatik digantikan oleh nilai eqivalennya dalam himpunan (0, 1, 2,..., M-1) atau (1, 2, 3,..., M).

Contoh:

Aritmatik Modulo 12

6 + 9 ≡ 3, didapat dari 6 + 9 = 15 – 12 = 3,

7 x 5 ≡ 11, didapat dari 7 x 5 = 35 – 12 = 23 – 12 = 11,

1 – 5 ≡ 8, didapat dari 1 – 5 = -4 + 12 = 8,

2 + 10 ≡ 0 ≡ 12, didapat dari 2 + 10 = 12, atau 2 + 10 = 12 – 12 = 0.

Fungsi Integer dan Nilai Absolut

Jika x adalah sembarang bilangan real. Nilai integer dari x (ditulis INT(x)) adalah mengubah nilai x menjadi suatu integer dengan cara menghilangkan bagian pecahan bilangan tersebut.

Contoh:

INT(3.14) = 3, INT(√5) = 2, INT(-8.5) = -8, INT(7) = 7

Perhatikan bahwa INT(x) = $\lfloor x \rfloor$ atau INT(x) = $\lceil x \rceil$ tergantung apakah x positif atau negatif. Nilai absolut dari sembarang bilangan real (ditulis ABS(x) atau |x| didefinisikan sebagai x atau –x terbesar. ABS(0) = 0 untuk $x \neq 0$, ABS(x) = x atau ABS(x) = -x tergantung apakah x positif atau negatif.

Contoh:

|-15| = 15, |7| = 7, |-3.33| = 3.33, |4.44| = 4.44, |0.0075| = 0.0075

Simbol Sums

Perhatikan suatu urutan a_1, a_2, a_3,... maka penjumlahan $a_1 + a_2 + \ldots + a_n$ dan $a_m + a_{m+1} + \ldots + a_n$ akan dinotasikan menjadi $\sum_{j=1}^{n} a_j$ dan $\sum_{j=m}^{n} a_j$.

Huruf j pada ekspresi di atas disebut dummy index atau dummy variable. Huruf lain yang biasa digunakan adalah i, k, s, dan t.

Contoh:

$\sum_{i=1}^{n} a_i b_i = a_1b_1 + a_2b_2 + \ldots + a_nb_n$

$\sum_{j=2}^{5} j^2 = 2^2 + 3^2 + 4^2 + 5^2 = 4 + 9 + 16 + 25 = 54$

$\sum_{j=1}^{n} j = 1 + 2 + \ldots + n$

Fungsi Faktorial

Perkalian bilangan integer positif dari 1 sampai n, di notasikan sebagai n! (baca: n faktorial).

n! = 1 * 2 * 3 *... (n-2)(n-1)n, dengan catatan 0! = 1.

Contoh:

a. 2! = 1 * 2 – 2; 3! = 1 * 2 * 3 = 6; 4! = 1 * 2 * 3 * 4 = 24
b. Untuk n > 1, n! = n * (n-1)!. Maka 5! = 5 * 4! = 5 * 24 = 120; 6! = 6 * 5! = 6 * 120 = 720.

Permutasi

Permutasi adalah suatu himpunan n elemen dengan susunan elemen tertentu. Sebagai contoh, permutasi dari himpunan yang mengandung elemen a, b, c adalah sebagai berikut:

abc, acb,
bac, bca,
cab, cba

Terdapat n! permutasi dari himpunan n elemen. Jadi ada 4! = 24 permutasi dari himpunan dengan 4 elemen, 5! = 120 permutasi dari himpunan dengan 5 elemen dan seterusnya.

Eksponen dan Logaritma

Perhatikan definisi berikut ini:

$a^m = a * a *... * a$ (m kali), $a^0 = 1$, $a^{-m} = 1/a^m$

Untuk pangkat m/n, maka $a^{m/n} = \sqrt[n]{a^m} = \left(\sqrt[n]{a}\right)^m$

Contoh:

$2^4 = 16$, $2^{-4} = 1/2^4 = 1/16$, $125^{2/3} = 5^2 = 25$

Misalkan b adalah bilangan positif, logaritma dari bilangan positif x berdasarkan basis b ditulis sebagai $\log_b x$

$y = \log_b x$ dan $b^y = x$

Contoh:

$\log_2 8 = 3$ karena $2^3 = 8$; $\log_{10} 100 = 2$ karena $10^2 = 100$
$\log_2 64 = 6$ karena $2^6 = 64$; $\log_{10} 0.001 = -3$ karena $10^{-3} = 0.001$

Lebih jauh, untuk sembarang basis b:

$\text{Log}_b\ 1 = 0$ karena $b^0 = 1$

$\text{Log}_b\ b = 1$ karena $b^1 = b$

3.2. Notasi Algoritma

Algoritma adalah suatu daftar langkah-langkah instruksi yang terdefinisikan dengan baik untuk memecahkan masalah tertentu.

Contoh:

Suatu array DATA mengandung nilai numerik. Misalkan kita ingin mencari lokasi LOC dan nilai MAX dari elemen terbesar dari Array DATA. Salah satu cara untuk memecahkan masalah tersebut adalah sebagai berikut:

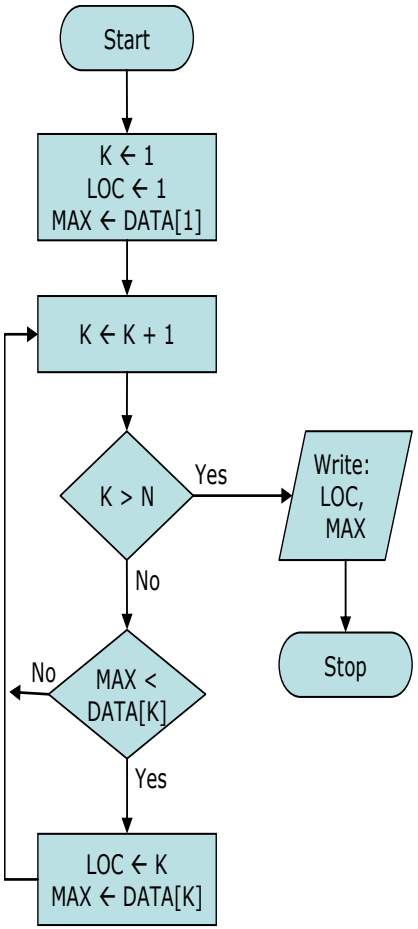
Dimulai dengan LOC = 1 dan MAX = DATA[1]. Kemudian bandingkan MAX dengan tiap elemen berikutnya DATA[K] dari DATA. Jika DATA[K] melebihi MAX, maka update LOC dan MAX sehingga LOC = K dan MAX = DATA[K]. Nilai akhir didapat dari LOC dan MAX yang menunjukkan lokasi dan nilai terbesar dari array DATA.

(Elemen terbesar dalam Array) Array DATA dengan n jumlah data. Algoritma ini mencari lokasi LOC dan nilai MAX dari elemen terbesar array DATA. Variabel K digunakan sebagai counter.

Step 1. [Inisialisasi] Set K:= 1, LOC:= 1 dan MAX:= Data[1]
Step 2. [Penambahan counter] Set K:= K + 1.
Step 3. [Tes Counter] if K > N then:
Write: LOC, MAX and Exit.
Step 4. [Bandingkan dan Update] If MAX < DATA[K] then:
Set LOC:= K dan MAX:= DATA[K].
Step 5. [Repeat Loop] Go To Step 2.

Secara formal, bentuk algoritma terdiri dari dua bagian. Bagian pertama adalah paragraf yang menjelaskan guna dari algoritma,

variabel yang ada dalam algoritma dan daftar input data. Bagian kedua dari algoritma terdiri dari daftar langkah-langkah yang akan dijalankan.



Gambar 3. 1. Contoh Algoritma

Berikut ini adalah aturan-aturan yang akan kita gunakan dalam algoritma.

Langkah, Kontrol dan Exit

Langkah-langkah dalam algoritma dijalankan berurutan mulai dari langkah pertama, kecuali dinyatakan lain. Kontrol dapat dialihkan ke Step n dari algoritma menggunakan pernyataan Go To Step n. Sebagai

contoh, pada algoritma di atas Step 5 mengalihkan kontrol kembali ke Step 2. Secara umum, pernyataan Go To akan digantikan dengan struktur kontrol yang akan kita bahas pada bagian lain.
Jika beberapa pernyataan muncul pada langkah yang sama seperti pada Step 1, maka pernyataan akan dijalankan dari kiri ke kanan.
Algoritma selesai jika ditemukan pernyataan Exit.

Komentar

Tiap langkah dapat mengandung komentar yang ditulis dalam kurung siku yang menjelaskan guna dari langkah tersebut. Komentar biasanya digunakan pada bagian awal dari tiap langkah.

Nama Variabel

Nama variabel ditulis menggunakan huruf besar, seperti MAX dan DATA. Variable dengan huruf tunggal digunakan sebagai counter atau subscript dan juga akan ditulis menggunakan huruf besar (contohnya K dan N).

Pernyataan Perintah

Statement perintah ditulis menggunakan notasi titik dua sama dengan seperti yang digunakan dalam Pascal. Sebagai contoh: MAX:= DATA[1], artinya memasukkan nilai DATA[1] ke dalam variabel MAX. Beberapa buku menggunakan notasi ← atau = untuk operasi ini.

Input dan Output

Data dapat diinput dan dimasukkan ke dalam variabel, dalam hal ini menggunakan pernyataan Read dengan format berikut:

Read: Nama Variabel

Pesan (diapit dengan tanda kutip) dan data dalam variabel dapat ditampilkan menggunakan pernyataan Write atau Print dengan format sebagai berikut:

Write: Pesan dan atau Nama Variabel.

Procedure

Istilah procedure digunakan untuk modul algoritma yang independen untuk memecahkan suatu masalah.

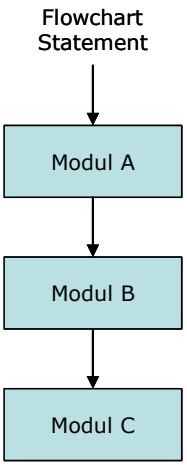
3.3. Stukrtur Kontrol

Algoritma dan program komputer akan lebih mudah dipahami jika dibuat dalam modul dan menggunakan alur atau kontrol logika yaitu:

a. *Sequence Logic* atau *sequential flow* (alur berurutan)
b. *Selection Logic* atau *conditional flow* (percabangan)
c. *Iteration Logic* atau *repetitive flow* (perulangan)

Sequence Logic

Jika tidak dinyatakan lain, modul dijalankan dalam urutan yang jelas. Urutan secara eksplisit ditampilkan dalam bentuk urutan langkah-langkah sesuai dengan urutan penulisan modul.



Gambar 3. 2. Sequence Logic

Selection Logic (Percabangan)

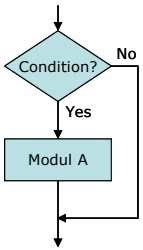
Selection logic melibatkan beberapa kondisi yang mengarahkan alur ke satu pilihan dari beberapa alternatif. Struktur yang menggunakan logika ini disebut struktur kondisional atau Struktur IF.

Struktur kondisional ini terdiri dari tiga jenis yaitu:

a. Single alternative

If <Condition>, then
    Modul A
[End of If Structure]

Jika kondisi terpenuhi (benar) maka Modul A yang mengandung satu atau lebih pernyataan akan dijalankan. Jika tidak maka Modul A kan dilewati dan kontrol berpindah ke langkah berikutnya dalam algoritma.

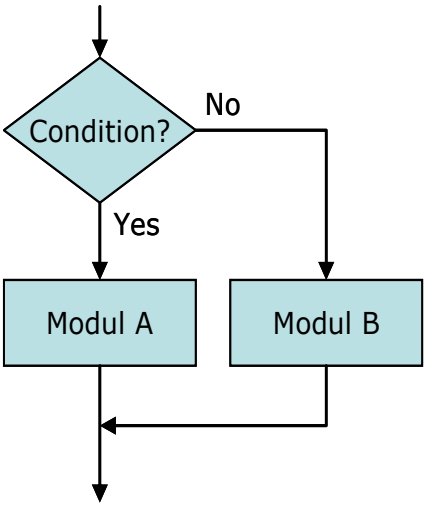


Gambar 3. 3. Single alternative

b. Double Alternative

If <Condition>, then
    Modul A
Else:
    Modul B
[End of If Structures]

Jika kondisi terpenuhi (benar) maka Modul A akan dijalankan, jika tidak maka Modul B yang akan dijalankan.



Gambar 3. 4. Double Alternative

c. Multiple Alternatives

If <Condition_1>, then:
    Modul A
Else If <Condition_2>, then:
    Modul B
Else If <Condition_n> then:
    Modul C
Else:
    Modul D
[End of If Structures]

Struktur logika di atas memungkinkan hanya satu modul yang akan dijalankan. Jika kondisi pertama terpenuhi maka Modul A akan dijalankan, jika tidak maka kondisi kedua akan diuji kebenarannya, demikian seterusnya.

Contoh:

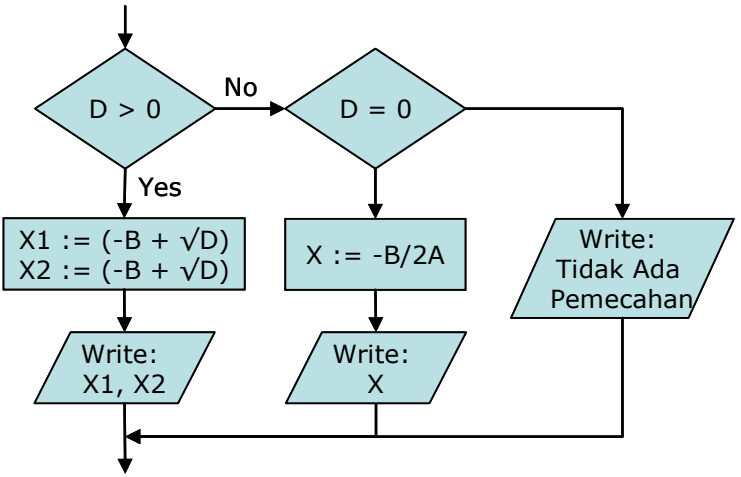
Persamaan Kuadrat, $ax^2 + bx + c = 0$, dimana $a \neq 0$.
x dapat dihitung dengan rumus:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Nilai D = b^2 – 4ac disebut sebagai Diskriminan. Jika D bernilai negatif, maka dipastikan tidak ada pemecahan, kemudian jika D = 0 maka ada satu pemecahan yaitu x = -b/2a. Jika D bernilai positif, maka ada dua pemecahan yang berbeda. Berikut ini adalah algoritma untu memecahkan masalah persamaan kuadrat.

(Persamaan Kuadrat) Algoritma ini memerlukan input koefisien A, B, C sebagai persamaan kuadrat dan menghasilkan pemecahan masalah jika ada.

Step 1. Read: A, B, C
Step 2. Set D:= B2 – 4AC
Step 3. If D > 0 then
  a. Set X1:= (-B + √D)/2A dan X2:= (-B - √D)/2A
  b. Write: X1, X2
  Else If D = 0, then:
  a. Set X:= -B/2A
  b. Write: X
  Else:
  Write: ’Tidak ada pemecahan’.
  [End of IF Structures]
Step 4. Exit.



Gambar 3. 5. Multiple Alternatives

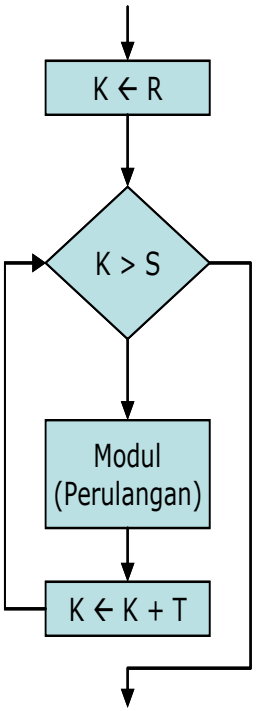
Iteration Logic (Perulangan)

Logika ini terdiri dari tiga struktur perulangan yaitu For... do, While... do dan Repeat... Until. Struktur For... do digunakan untuk perulangan dengan jumlah pasti, sedangkan struktur While... do dan Repeat... Until digunakan untuk perulangan dengan jumlah tidak pasti.

a. For

```
For K:= R to S by T:
        [Module]
[End of Loop]
```

Variabel R disebut nilai awal, S adalah nilai akhir dan T adalah pertambahan. Perhatikan bahwa perulangan akan dilakukan dengan K = R, kemudian K = R + T, kemudian K = R + 2T dan seterusnya. Perulangan berakhir apabila nilai K > S.

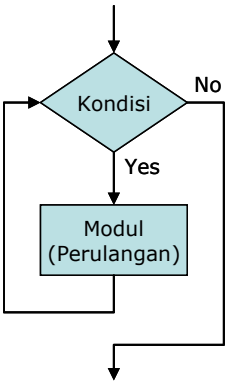


Gambar 3. 6. For

b. While... do

While <condition> do
    [Modul]
[End of Loop]

Perulangan berlangsung selama kondisi terpenuhi dan akan berakhir jika kondisi bernilai false. Perlu ditekankan bahwa harus ada pernyataan inisialisasi kondisi sebelum struktur perulangan yang akan mengontrol perulangan, dan harus ada pernyataan dalam struktur perulangan yang mengubah kondisi tersebut.



Gambar 3. 7. While … Do

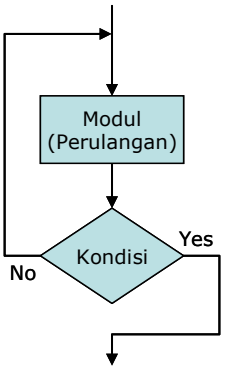
c. Repeat... Until

Repeat

[Modul]

Until <Condition>

Perhatikan, perbedaan struktur perulangan While.. do dengan Repeat.. Until, terletak pada posisi kondisi. Pada struktur While... do, kondisi terletak pada bagian awal struktur, sedangkan struktur Repeat... Until, kondisi terletak pada akhir struktur. Ini berarti pada struktur Repeat... Until, perulangan minimal dilakukan satu kali.



Gambar 3. 8. Repeat … Until

Contoh:

Algoritma berikut ini merupakan penyempurnaan **Error! Reference source not found.** dengan menggunakan struktur While... Do

Algoritma: (Elemen terbesar dalam Array) Array DATA dengan n jumlah data. Algoritma ini mencari lokasi LOC dan nilai MAX dari elemen terbesar array DATA. Variabel K digunakan sebagai counter.

Step 1. [Inisialisasi] Set K:= 1, LOC:= 1 dan MAX:= Data[1]
Step 2. Repeat Step 3 and 4 While K ≤ N:
Step 3. If MAX < DATA[K], then:
Set LOC:= K and MAX:= DATA[K].
[End of If Structure]
Step 4. Set K:= K + 1.
[End of Step 2 loop]
Step 5. Write: LOC, MAX
Step 6. Exit.

3.4. Kompleksitas Algoritma

Analisa algoritma merupakan kegiatan utama dalam ilmu computer. Untuk membandingkan suatu algoritma, kita harus mempunyai beberapa kriteria untuk mengukur efisiensi dari algoritma kita.

Misalkan M adalah suatu algoritma dan b adalah jumlah ukuran input datanya. Waktu dan ruang yang digunakan oleh algoritma M adalah dua hal utama untuk mengukur efisiensi dari M. Waktu diukur dengan cara menghitung berapa banyak operasi kunci yang dilakukan, misalnya banyaknya jumlah perbandingan dalam algoritma pengurutan dan pencarian. Ruang diukur berdasarkan jumlah maksimum memory yang dibutuhkan oleh algoritma.

Kompleksitas Algoritma M adalah fungsi f(n) yang memberikan waktu untuk menjalankan dan atau banyaknya ruang penyimpanan yang dibutuhkan algoritma dalam hal jumlah input data.

Contoh berikut ini menggambarkan bahwa fungsi f(n) yang memberikan nilai waktu dari suatu algoritma tidak hanya tergantung dari ukuran n input data tapi juga berdasarkan data tertentu.

Contoh:

Misalkan kita mempunyai suatu cerita pendek TEXT, dan kita ingin mencari dalam TEXT tersebut untuk mencari keberadaan pertama kata dengan tiga huruf W. Jika W adalah kata dengan tiga huruf ’the’ maka sepertinya W muncul dekat dengan bagian awal dari TEXT, jadi nilai f(n) akan kecil. Di lain pihak, jika W adalah kata dengan tiga huruf ’zoo’ maka W bisa jadi tidak ada dalam TEXT, jadi nilai f(n) akan besar.

Dari diskusi diatas memandu kita pada pertanyaan untuk mencari kompleksitas fungsi f(n) untuk beberapa hal. Dua hal utama yang biasanya diselidiki dalam teori kompleksitas adalah:

a. *Worst case*: nilai maksimun dari fungsi f(n) untuk sembarang input.
b. Average Case: nilai yang diinginkan dari fungsi f(n).

Kadang kita juga harus mempertimbangkan nilai kemungkinan minimal dari fungsi f(n) yang biasa disebut dengan Best Case.

Contoh:

Misalkan suatu array linear DATA mengandung n elemen dan suatu ITEM informasi. Kita ingin mencari lokasi LOC dari ITEM dalam array DATA, atau mengirimkan pesan seperti LOC = 0 yang menandakan ITEM yang dicari tidak terdapat dalam DATA. Algoritma pencarian Linear (linear search) memecahkan masalah ini dengan membandingkan ITEM, satu per satu dengan elemen dalam DATA. Kita membandingkan ITEM dengan DATA[1], kemudian DATA[2]

dan seterusnya, sampai kita menemukan LOC dimana ITEM = DATA[LOC]. Algoritma tersebut disajikan sebagai berikut:

Algoritma: (Linear Search) Suatu array linear DATA dengan N elemen dan ITEM informasi. Algoritma ini mencari lokasi LOC dari ITEM dalam array DATA atau menyatakan LOC = 0.

Step 1. [Inisialisasi] Set K:= 1 and LOC:= 0.
Step 2. Repeat Step 3 dan 4 While LOC = 0 and K ≤ N.
Step 3. If ITEM = DATA[K], then Set LOC:= K.
Step 4. Set K:= K + 1 [Penambahan Counter]
[End of Step 2 Loop]
Step 5. [Sukses?]
If LOC = 0, then
Write: 'ITEM tidak ada dalam DATA'.
Else:
Write: 'LOC adalah lokasi ITEM'.
[end of If Structure]
Step 6. Exit.

Kompleksitas dari algoritma di atas ditunjukkan oleh banyak C perbandingan antara ITEM dan DATA[K]. Kita akan mencari C(n) untuk *Worst case* dan Average Case.

Worst case

Jelas bahwa *worst case* ada apabila ITEM adalah elemen terakhir dalam Array DATA atau tidak ada dalam array tersebut. Dalam situasi

di atas C(n) = n. Maka, C(n) = n adalah kompleksitas worst-case pada algoritma pencarian linear.

Average Case

Disini kita asumsikan bahwa ITEM ada dalam DATA, dan ITEM tersebut ada pada sembarang posisi dalam DATA. Maka, jumlah perbandingan dapat berupa sembarang angka 1, 2, 3,..., n dan tiap angka mempunyai probabilitas kemunculan p = 1/n. Kemudian

$$C(n) = 1*\frac{1}{n} + 2*\frac{1}{n} + \ldots + n*\frac{1}{n}, \quad = (1+2+\ldots+n)*\frac{1}{n}, \quad = \frac{n(n+1)}{2}*\frac{1}{n} = \frac{n+1}{2}.$$

Ini selaras dengan rasa intuitive bahwa rata-rata jumlah perbandingan yang diperlukan untuk mencari lokasi ITEM adalah kira-kira sama dengan setengah jumlah elemen dalam DATA.

3.5. Subalgoritma

Subalgoritma adalah suatu modul algoritma lengkap dan independen yang digunakan (dipanggil) oleh algoritma utama atau oleh subalgoritma lainnya. Suatu subalgoritma menerima nilai, yang disebut argumen, dari algoritma yang memanggilnya, melakukan perhitungan dan mengembalikan hasil ke algoritma yang memanggilnya. Subalgoritma didefinisikan secara independen jadi subagoritma ini dapat dipanggil oleh algoritma lain yang berbeda atau dipanggil beberapa kali dalam algoritma yang sama. Hubungan antara algoritma dan subalgoritma serupa dengan hubungan antara program utama dan subprogram dalam bahasa pemrograman.

Perbedaan utama antara bentuk subalgoritma dengan algoritma adalah subalgoritma biasanya mempunyai bentuk heading seperti NAME(PAR_1, PAR_2,..., Par_k).

NAME menunjukkan nama subalgoritma yang digunakan untuk memanggil, dan PAR_1, PAR_2,..., Par_k menunjukkan parameter yang

akan digunakan untuk mengirim data antara subalgoritma dengan algoritma yang memanggilnya.

Perbedaan lain adalah subalgoritma mengandung pernyataan Return bukan Exit seperti pada algoritma. Salah satu perbedaan antar subalgoritma adalah bahwa function subalgoritma hanya mengembalikan satu nilai ke algoritma yang memanggilnya, sedangkan procedure subalgoritma dapat mengirimkan satu atau lebih nilai.

Contoh:

MEAN Function subalgoritma berikut ini digunakan untuk mencari rata-rata AVE dari tiga angka A, B dan C.

Function MEAN(A, B, C)

1. Set AVE:= (A + B + C)/3
2. Return(AVE)

MEAN adalah nama function subalgoritma dan A, B, dan C adalah parameternya. Pernyataan Return, termasuk variable AVE dalam tanda kurung adalah nilai yang akan dikembalikan ke program pemanggilnya.

Subalgoritma MEAN digunakan dalam algoritma sama halnya dengan penggunaan function subprogram yang dipanggil oleh program. Sebagai contoh, misalkan suatu algoritma mengandung penyataan Set TEST:= MEAN(T_1, T_2, T_3) dimana T_1, T_2, T_3 adalah nilai tes. Nilai argumen T_1, T_2, T_3 dikirim ke parameter A, B, dan C dalam subalgoritma, subalgoritma MEAN dijalankan dan kemudian nilai AVE dikembalikan ke program dan menggantikan MEAN(T_1, T_2, T_3) dalam pernyataan. Akhirnya rata-rata nilai T_1, T_2, dan T_3 akan dimasukkan ke dalam variabel TEST.

Contoh:

Berikut ini adalah procedure SWITCH yang akan menukar nilai AAA dan BBB.

Procedure: SWITCH(AAA, BBB)

1. Set TEMP:= AAA, AAA:= BBB dan BBB:= TEMP.

2. Return

Procedure dapat dipanggil menggunakan pernyataan Call. Contoh penggunaan pernyataan Call: Call SWITCH(BEG, AUX). Pernyataan tersebut akan memberikan efek saling tukar menukar nilai antara BEG dan AUX. Secara khusus, saat procedure SWITCH dipanggil, argumen BEG dan AUX akan dikirim masing-masing ke parameter AAA dan BBB, procedure dijalankan, dimana akan menukarkan nilai AAA dan BBB dan kemudian nilai baru AAA dan BBB akan dikirimkan kembali masing-masing ke BEG dan AUX.

Catatan: sembarang function subalgoritma dapat dengan mudah diterjemahkan ke dalam bentuk procedure subalgoritma dengan jalan menggabungkan parameter tambahan yang digunakan untuk mengembalikan nilai yang dihitung ke algoritma pemanggilnya. Sebagai contoh, function subalgoritma MEAN jika diubah ke dalam bentuk procedure subalgritma akan menjadi MEAN(A, B, C, AVE), dimana paramater AVE akan diisi dengan rata-rata A, B, C. Dengan demikian pernyataan: Call MEAN(T_1, T_2, T_3, TEST) akan mengirimkan rata rata nilai T_1, T_2, T_3 ke TEST.

3.6. Variabel

Tiap variabel dalam suatu algoritma atau program harus mempunyai tipe data yang menentukan kode yag akan digunakan untuk menyimpan nilainya. Berikut ini adalah 4 tipe data yang biasa digunakan:

a. Karakter. Data dikodekan menggunakan beberapa kode karakter seperti EBCDIC atau ASCII. 8 bit kode EBCDIC ada pada tabel di bawah ini. Satu karakter biasanya disimpan dalam satu byte.

Tabel 3. 1. Karakter

Char	Zone Numeric	Hex	Char	Zone Numeric	Hex	Char	Zone Numeric	Hex
A	1100 0001	C1	S	1110 0010	E2	Blank	0100 0000	40
B	1100 0010	C2	T	1110 0011	E3	.	0100 1011	4B

C								
D								

b. Real (atau floating point). Data numerik dikodekan menggunakan bentuk eksponensial dari data.
c. Integer (atau fixed point). Integer positif dikodekan menggunakan bentuk binary dan integer negatif menggunakan beberapa variasi binary seperti komplimen kedua.
d. Logical. Variabel hanya mempunyai nilai True atau False, karenanya tipe ini dapat dikodekan dengan hanya menggunakan satu byte, 1 for True dan 0 untuk False. Kadang-kadang bytes 1111 1111 dan 0000 0000 digunakakn untuk true dan false.

Contoh:

Misalkan pada komputer dengan memory 32-bit, lokasi X mengandung urutan bits sebagai berikut: 0110 1100 1100 0111 1101 0110 0110 1100

Tidak ada cara lain untuk mengetahui isi dari sel tersebut kecuali kita tahu type data X tersebut.

a. Misalkan X dideklarasikan sebagai tipe karakter EBCDIC, maka karakter yang tersimpan dalam X adalah %GO%.
b. Misalkan X dideklarasikan sebagi tipe integer atau real, maka X akan berisi suatu bilangan integer atau bilangan real.

Variabel Lokal dan Global

Program komputer diorganisasikan ke dalam program utama dan beberapa subprogram yang menuntun kita pada variabel lokal dan global. Biasanya, tiap modul program mengandung daftar variabelnya sendiri yang disebut dengan lokal variabel, yang hanya dapat diakses oleh modul program tersebut. Demikian juga dengan modul subprogram yang mengandung parameter, variabel yang mengirimkan data antara subprogram dengan program yang memanggilnya.

Contoh:

Perhatikan procedure SWITCH(AAA, BBB) pada contoh Contoh. Variabel AAA dan BBBB adalah parameter, mereka digunaka untuk mengirimkan data antara procedure dan program pemanggilnya. Disisi lain, variable TEMP dalam procedure tersebut disebut dengan variabel lokal. Variabel ini hanya berfungsi dalam procedure, dalam hal ini variabel hanya bisa diakses dan diubah pada saat procedure dijalankan. Sesungguhnya, nama TEMP boleh digunakan untuk nama variabel pada modul lain dan penggunaan nama tersebut tidak bertentangan dengan eksekusi procedure SWITCH.

Perancang bahasa pemrograman menyadari bahwa akan lebih mudah jika terdapat variabel yang dapat diakses oleh beberapa atau bahkan semua modul program dalam program komputer. Variabel yang dapat diakses oleh beberapa modul program disebut Non-Lokal variabel.

Maka, ada dua cara dasar berkomunikasi antar modul, yaitu:

a. *Directly*, menggunakan paramater yang terdefinisi dengan baik.
b. *Indirectly*, menggunakan variabel lokal dan global.

Perubahan secara tidak langsung dari nilai suatu variabel dalam suatu modul oleh modul lain disebut efek samping (*side effect*).

Pada penelitian yang berjudul "Analisis Kualitas Pelayanan Pendidikan Dengan Menggunakan Metode Importance Performance Analysis (IPA) Berbasis Web" merupakan sistem pengukuran kepuasan pelayanan terhadap mahasiswa. Penelitian ini menggunakan instrumen SERVQUAL untuk mengukur kepuasan mahasiswa dan analisis data dilakukan dengaan metode importance performance analysis (IPA) dan pengujian dengan membuat suatu rancangan sistem kuisioner berbasis web yang mana harapannya sistem ini dapat mendukung pengambilan keputusan perguruan tinggi khususnya Prodi Komputerisasi Akuntansi Politeknik Negeri Banjarmasin.

Aplikasi yang telah dilakukan pengujiannya menampilkan halaman layanan kuisioner yang berisikan beberapa pertanyaan yang telah dibuat pada system database, dimana setiap pertanyaan terdiri dari pertanyaan dari kepentingan dan beberapa pertanyaan dari kepentingan dan kinerja dari bukti fisik begitu juga untuk kuisioner kehandalan, daya tanggap, jaminan, dan empaty hampir sama.

Pada penelitian ini didapat hasil dari keputusan dengan metode IPA berdasarkan variabel yang digunakan sebagai berikut:

1. Kuadran I (Prioritas Utama)

   Atribut yang berada pada kuadran ini dianggap sangat penting oleh mahasiswa tetapi kualitas pelayanan tidak memuaskan. Atribut atribut ini menjadi prioritas utama untuk segera dilakukan perbaikan oleh pihak manajemen kampus khususnya prodi komputerisasi akuntansi jurusan akuntansi. Adapun atribut-atributnya sebagai berikut;

   A3 : Kondisi peralatan pratikum di laboratorium
   A5 : Kelengkapan buku, jurnal dan literature kuliah di perpustakaan
   A6 : Kelengkapan peralatan pratikum di laboratorium
   A23 : Pembahasan ulang soal ujian oleh dosen dan asisten lab.

2. Kuadran II (Pertahankan Prestasi)

   Atribut yang berada pada kuadran ini dianggap sangat penting oleh mahasiswa dan kualitas pelayanannya sangat memuaskan. Mahasiswa sangat puas dengan atributatribut yang berada pada kuadran ini, pihak manajemen kampus prodi komputerisasi akuntansi jurusan Akuntansi dapat mempertahankan kualitas pelayanan atribut-atribut yang berada pada kuadran ini. Adapun atribut-atributnya sebagai berikut:

   A1 : Kenyamanan Ruangan kuliah (kursi, kipas angin, AC, kebersihan, dsb)

A2 : Ketersediaan kawasan koneksi internet (WiFi) yang mendukung kegiatan belajar
A4 : Kebersihan dan tersedianya air bersih di toilet/kamar mandi
A9 : Dosen/Asisten dosen/asisten lab menyampaikan materi belajar dengan jelas
A12 : Soal-soal ujian sesuai dengan materi yang diberikan
A13 : Sistem pemberian nilai dilakukan secara objektif
A16 : Ketersediaan pihak jurusan dalam merespon dan menanggapi keluhan mahasiswa
A18 : Keamanan lingkungan kampus dan adanya petugas keamanan
A19 : Kemudahan dalam memperoleh informasi tentang sistem Pendidikan (kurikulum, jadwal kuliah, beasiswa, dll)
A21 : Dosen memberikan motivasi bagi mahasiswa saat sedang mengajar dikelas
A24 : Adanya organisasi kampus penunjang kegiatan mahasiswa

3. Kuadaran III (Prioritas Rendah)

Atribut yang berada pada kuadran ini dianggap tidak terlalu penting oleh mahasiswa dan kualitas pelayanannya kurang memuaskan. Pihak manajemen kampus sebaiknya meningkatkan lagi kualitas pelayanan pada kuadran ini, Adapun atributatributnya sebagai berikut:

A8 : Kemudahan pencarian buku diperpustakaan
A10 : Dosen mengajar sesuai dengan jadwal yang ditetapkan
A11 : Pengawas ujian memulai dan mengakhiri ujian dengan tepat waktu

A20 : Kejelasan adanya biaya administrasi dan jumlah biaya dalam pengurusan berkas

A22 : Adanya dosen pengganti ketika dosen berhalangan mengajar

4. Kuadran IV (Berlebihan)

   Atribut yang berada pada kuadran ini dianggap tidak terlalu penting oleh mahasiswa dan kualitas pelayanannya memuaskan. Mahasiswa sudah merasa sangat puas atas kualitas pelayanan pendidikan pada kuadran ini. Adapun atribut-atributnya sebagai berikut:

   A7 : Ketersediaan tempat parkir yang memadai

   A14 : Sistem pembayaran uang kuliah

   A15 : Kemampuan dosen dalam menjawab pertanyaan mahasiswa secara jelas

   A17 : Staff melayani pengurusan berkas secara cepat dan professional

3.7. Soal Latihan

1. Tentukan:
   a. $\lfloor 7.5 \rfloor, \lfloor -7.5 \rfloor, \lfloor -18 \rfloor, \lfloor \sqrt{30} \rfloor, \lfloor \sqrt[3]{30} \rfloor, \lfloor \pi \rfloor$
   b. $\lceil 7.5 \rceil, \lceil -7.5 \rceil, \lceil -18 \rceil, \lceil \sqrt{30} \rceil, \lceil \sqrt[3]{30} \rceil, \lceil \pi \rceil$
2. Tentukan:
   a. 26 (mod 7), 34 (mod 8), 2345 (mod 6), 495 (mod 11).
   b. -26 (mod 7), -2345 (mod 6), -371 (mod 8) dan -39 (mod 3).
   c. Menggunakan aritmatik modulo 15, hitung 9 + 13, 7 + 11, 4 – 9 dan 2 – 10.
3. Buat daftar permutasi menggunakan angka 1, 2, 3, 4
4. Tentukan:
   a. $2^{-5}, 8^{2/3}, 25^{-3/2}$
   b. $\text{Log}_2\ 32, \log_{10} 1000, \log_2 (1/16)$
   c. $\lfloor \log_2 1000 \rfloor, \lfloor \log_2 0.01 \rfloor$

5. Perhatikan **Error! Reference source not found.** yang mencari location LOC dan nilai MAX elemen terbesar dalam array DATA dengan n elemen. Perhatikan kompleksitas fungction C(n), yang mengukur berapa kali LOC dan MAX diupdate pada step 3.
    a. Jelaskan dan cari C(n) untuk *worst case*.
    b. Jelaskan dan cari C(n) untuk best case.
    c. Cari C(n) untuk average case jika n = 3, asumsikan susunan elemen dalam DATA sama.

BAB IV

PEMROSESAN STRING

Capaian Pembelajaran:

1. Mampu menjelaskan pemrosesan string mulai dari istilah-istilah dasar
2. Mampu melakukan penyimpanan string.
3. Mampu menjelaskan type data karakter, operasi string, pengolahan kata serta pattern matching algorithms.

4.1. Pengertian

Tiap bahasa pemrograman mempunyai karakter set sendiri yang digunakan untuk berkomunikasi dengan komputer. Set ini biasanya terdiri dari:

```
Alfabet          :    A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
Digit            :    1 2 3 4 5 6 7 8 9
Karakter Khusus  :    + - / * ( ),. $ = ' □
(blank space)
```

Rangkaian terbatas S dengan 0 atau lebih karakter disebut sebagai string. Jumlah karakter dalam string disebut length. String dengan 0 karakter disebut empty string atau null string. String dinotasikan dengan mengapit karakternya menggunakan tanda kutip tunggal. Tanda tersebut disebut dengan string delimiters. 'THE END', 'TO BE OR NOT TO BE', '', '□□' adalah string dengan panjang (length) masing-masing 7, 8, 0 dan 2. Perlu ditekankan disini adalah blank space adalah karakter yang berarti memberikan kontribusi pada panjang string.

Misalkan S_1 dan S_2 adalah suatu string. String yang terdiri dari karakter-karakter S_1 diikuti oleh karakter-karakter S_2 disebut concatenation dari S_1 dan S_2, ini akan dinotasikan sebagai $S_1//S_2$. Sebagai contoh, 'THE'//'END' = 'THEEND' dan 'THE'//'□'//'END' = 'THE END'. Jelas bahwa panjang $S_1//S_2$ adalah penjumlahan panjang S_1 dan S_2.

Suatu string Y disebut substring dari string S jika string Y ada dalam string S, S = X//Y//Z. Jika X adalah empty string, maka Y disebut initial substring dari S, dan jika Z adalah empty string maka Y disebut terminal substring dari S. Contoh:

```
'BE OR NOT TO BE' adalah substring dari 'TO BE
OR NOT TO BE'
'THE' adalah initial substring dari 'THE END'.
```

Jelas bahwa jika Y adalah substring dari S, maka panjang Y tidak boleh melebihi panjang S.

Catatan: Karakter disimpan dalam computer menggunakan kode 6-bit, 7-bit atau 8-bit. Unit yang sesuai dengan jumlah bit yang diperlukan untuk mewakili suatu karakter disebut byte. Kecuali dinyatakan lain, satu byte biasanya terdiri dari 8 bit.

4.2. Penyimpanan String

Secara umum, string disimpan dalam 3 tipe struktur yaitu:

1. *Fixed-Length Structures*
2. *Variable-Length Structures*
3. *Linked Structure*

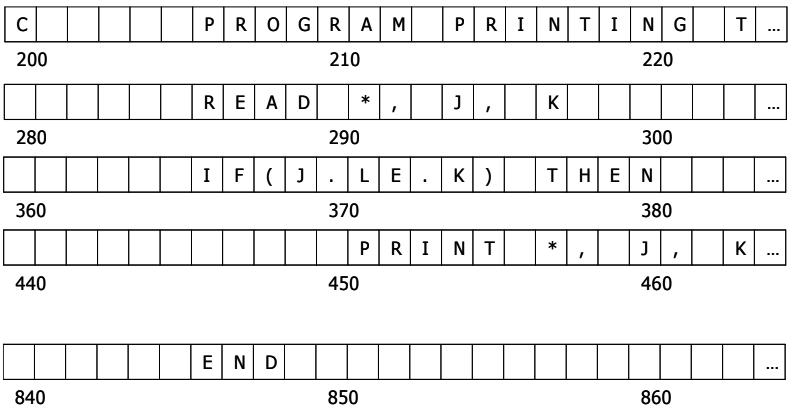
Record Oriented, Fixed-Length Storage

Dalam *fixed-length storage* tiap baris cetakan ditampilkan sebagai suatu record, dimana setiap record mempunyai panjang yang sama. Karena biasanya data dimasukkan menggunakan terminal dengan jumlah kolom 80, kita akan asumsikan bahwa record mempunyai panjang 80.

Contoh:

Misalkan suatu input dalam program FORTRAN. Menggunakan struktur record oriented, media fixed-length storage, input data akan ada di memory seperti gambar di bawah, disini kita asumsikan bahwa 200 adalah alamat dari karakter pertama program.

```
PROGRAM PRINTING TWO INTEGERS IN INCREASING
ORDER
      READ *, J, K
      IF(J.LE.K) THEN
            PRINT *, J, K
      ELSE
            PRINT *, K, J
      ENDIF
      STOP
      END
```



Gambar 4. 1. Record Oriented

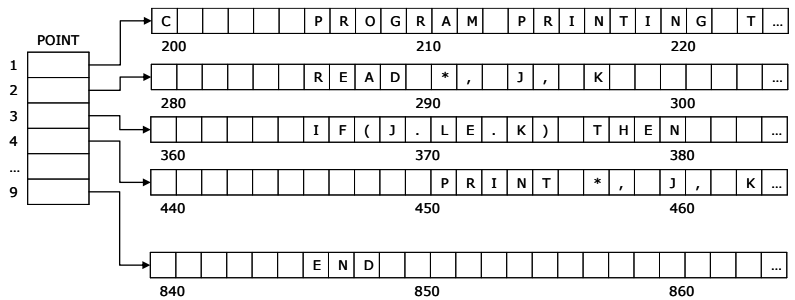
Keuntungan penyimpanan string menggunakan cara di atas adalah:

1. Kemudahan dalam pengaksesan data dari record tertentu.
2. Kemudahan dalam update data dalam record tertentu (sepanjang panjang data baru tidak melebihi panjang record).

Kerugian:

1. Waktu terbuang untuk membaca seluruh record jika penyimpanan banyak mengandung blank space yang tidak perlu.
2. Record tertentu memerlukan ruang lebih dari yang tersedia.
3. ketika koreksi terdiri dari lebih atau lebih sedikit karakter dibanding teks yang asli, mengubah kesalahan eja saja membutuhkan perubahan seluruh record.

Catatan: Misalkan kita ingin menyisipkan satu record baru. Hal ini memerlukan penggeseran record berikutnya ke lokasi baru dalam memory. Bagaimanapun, kerugian ini dapat diperbaiki seperti yang ditunjukkan pada gambar. Kita dapat menggunakan linear array POINT yang memberikan alamat dari tiap record berikutnya, jadi record tidak harus disimpan dalam lokasi yang berurutan dalam memory. Maka, menyisipkan record baru hanya memerlukan update pada array POINT.



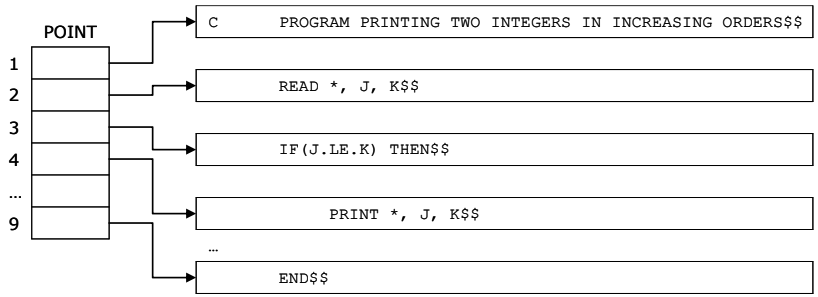
Gambar 4. 2. Record Oriented

Variable-Length Storage dengan Fixed Maximum

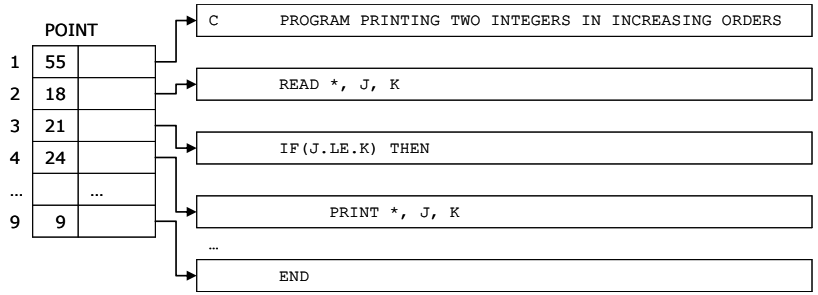
Walaupun string dapat disimpan menggunakan cara di atas, ada beberapa keuntungan kalau kita mengetahui panjang sebenarnya dari tiap string. Sebagai contoh, kita tidak harus membaca seluruh record jika string hanya memakai bagian awal dari lokasi memory.

Penyimpanan ini dapat dilakukan dengan dua cara yaitu ;

1. Menggunakan penanda, misalkan tanda $$, untuk menandai akhir dari string.
2. Membuat daftar panjang string sebagai tambahan item pada array pointer.



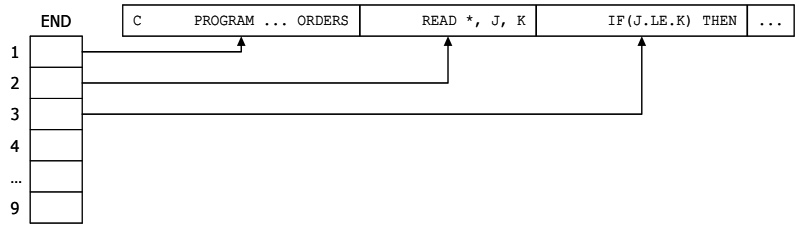
Gambar 4. 3. Variable-Length Storage



Gambar 4. 4. Variable-Length Storage

Catatan: Kita juga bisa mencoba menempatkan string berurutan satu dengan lainnya menggunakan tanda pemisah seperti $$ atau menggunakan pointer untuk menunjukkan lokasi dari string. Cara penyimpanan ini secara jelas menghemat tempat dan kadang-kadang digunakan dalam memory sekunder dimana record relatif lebih permanen dan memerlukan sedikit perubahan. Bagaimanapun, metode penyimpanan menjadi tidak efisien jika string dan panjangnya sering berubah-ubah.

```
C       PROGRAM ... ORDERS$$       READ *, J, K$$       IF(J.LE.K) THEN$$ ...
```

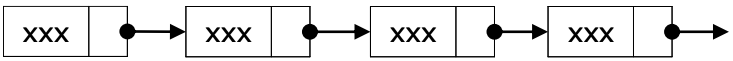


Gambar 4. 5. Variable-Length Storage

Linked Storage

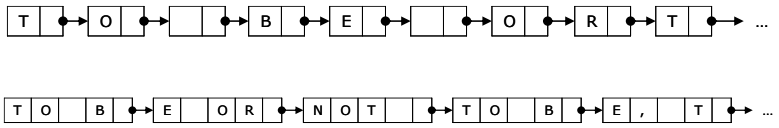
Komputer saat ini sering digunakan untuk pengolahan kata, seperti memasukkan, mengolah dan mencetak sesuatu. Oleh karena itu, komputer harus mampu memperbaiki dan mengubah, dalam hal ini menghapus, mengubah dan menyisipkan kata, frase, kalimat bahkan paragraf dalam teks. Bagaimanapun, cara yang didiskusikan diatas tidak dapat dengan mudah diterapkan untuk operasi ini. Maka, pada kebanyakan aplikasi pengolah kata, string disimpan dalam bentuk linked list.

Menggunakan *one-way linked list*, kita menggambarkan urutan sel memory yang disebut nodes, dimana tiap node terdiri dari item, yang disebut *link*, yang menunjuk ke node berikutnya dalam list (dalam hal ini berisi alamat node berikutnya).



Gambar 4. 6. *One-Way Linked List*

String dapat disimpan dalam link list. Tiap sel memory dicadangkan untuk satu karakter atau beberapa karakter, dan link yang berisi alamat cell yang mengandung karakter berikutnya atau kumpulan karakter dalam string. Gambar berikut menunjukkan bagaimana string ada di memory dengan satu karakter per node, dan **Error! Reference source not found.** menunjukkan bagaimana string berada di memory dengan 4 karakter per node.



Gambar 4. 7. Linked Storage

4.3. Tipe Data Karakter

Bagian berikut ini memberikan gambaran berbagai cara bahasa pemrograman menangani type data karakter. Seperti yang dijelaskan di depan, tiap tipe data mempunyai formula sendiri untuk menterjemahkan suatu urutan bit dalam memory.

Konstanta

Beberapa bahasa pemrograman menyatakan konstanta string dengan menempatkan string diantara tanda kutip atau tanda kutip dua. Sebagai contoh, 'THE END' dan "TO BE OR NOT TO BE" adalah suatu konstanta string dengan panjang masing-masing 7 dan 18.

Variabel

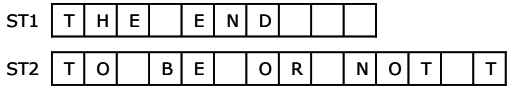
Tiap bahasa pemrograman mempunyai aturan sendiri dalam membentuk variable karakter. Bagaimanapun, variable dibagi menjadi tiga kategori: statis, semistatis dan dinamis. Menggunakan variabel karakter statis berarti variable dimana panjangnya didefinisikan sebelum program dijalankan dan tidak dapat diubah selama program berjalan. Menggunakan variabel karakter semistatis berarti variabel yang mempunyai panjang yang berbeda pada saat program sedang berjalan selama panjang tidak melebihi panjang minimum yang ditentukan sebelum program dijalankan. Menggunakan variabel karakter dinamis berarti suatu variabel yang mempunyai panjang yang dapat diubah selama program dijalankan.

Contoh:

a. Amati potongan program Pascal berikut ini:

```
Var
   ST1: String[10];
   ST2: String[14];
Begin
   ST1:= 'THE END';
   ST2:= 'TO BE OR NOT TO BE';
End.
```

Baris 1 s/d 3 merupakan bagian deklarasi variabel string dimana ST1 dan ST2 mempunyai panjang masing 10 dan 14. Setelah program dijalankan ST1 dan ST2 ada di memory dengan bentuk sebagai berikut:



Gambar 4. 8. Tipe Data Karakter

b. Dalam Pascal, string adalah kumpulan karakter (Char). Berikut adalah contoh deklarasi string menggunakan array karakter.

```
Var
   ST1: Array[1..20] of Char;
```

Contoh di atas adalah mendeklarasikan variabel ST1 dengan panjang 20 karakter. Lebih jauh ST1[1] adalah karakter pertama dalam ST1, ST1[2] adalah karakter kedua dan seterusnya.

4.4. Operasi String

Walaupun string secara sederhana digambarkan sebagai suatu linear array dari karakter, ada perbedaan mendasar dalam penggunaannya dibandingkan dengan tipe array lainnya. Secara khusus, kumpulan elemen yang berurutan dalam string (seperti kata, frase atau kalimat) disebut substring. Lebih jauh, unit dasar yang digunakan untuk mengakses string adalah substring, bukan karakter satuan.

Perhatikan contoh string berikut: ’TO BE OR NOT TO BE’. Kita dapat memandang string tersebut sebagai suatu urutan 18 karakter T, O, □, B dan seterusnya. Akan tetapi, karakter satuan tersebut tidak memiliki arti dibandingkan dengan substring ’TO’, ’BE’, ’OR’ dan lain-lain yang memiliki arti tersendiri.

Di lain sisi, perhatikan array integer dengan 18 elemen berikut ini: 4, 8, 6, 15, 9, 5, 4, 13, 8, 5, 11, 9, 9, 13, 7, 10, 6, 11. Unit dasar untuk mengakses array integer adalah elemen individu. Karena alasan di atas, berbagai operasi string telah dikembangkan yang secara normal tidak digunakan untuk tipe array lainnya. Bagian berikut membahas operasi yang berkaitan dengan string yang biasanya di pakai dalam pengolahan kata.

Substring

Untuk mengakses substring dari suatu string, ada tiga hal yang harus diketahui yaitu:

a. Nama string yang bersangkutan.
b. Posisi karakter pertama dari substring di dalam string.
c. Panjang substring atau posisi karakter terakhir dari substring dalam string.

Secara khusus perintah ini dapat ditulis dengan SUBSTRING(String, Awal, Panjang).

Contoh:

a. Menggunakan fungsi di atas kita mengetahui:

```
SUBSTRING('TO BE OR NOT TO BE', 4, 7) = 'BE
OR N'
SUBSTRING('THE END', 4, 4) = ' END'
```

b. Fungsi SUBSTRING(S, 4, 7) dinotasikan dalam bahasa pemrograman adalah sebagai berikut:

```
Pascal      COPY(S, 4, 7)
Basic       MID$(S, 4, 7)
```

Indexing

Indexing atau disebut juga pattern matching adalah mencari posisi dimana pola P pertama kali ditemukan dalam string T. Kita sebut operasi ini INDEX, dan ditulis sebagai INDEX(text, pola).

Jika pola P tidak terdapat dalam teks T, maka INDEX menghasilkan nilai 0. Argumen text dan pola dapat berupa konstanta string atau variabel string

Contoh:

a. Misalkan T mengandung teks: 'HIS FATHER IS THE PROFESSOR'. Maka:
   - `INDEX(T, 'THE') = 7`
   - `INDEX(T, 'THEN') = 0`
   - `INDEX(T, ' THE ') = 14`

b. Fungsi INDEX(Teks, Pola) dinotasikan dalam bahasa pemrograman:
   `Pascal  POS(Pola, Teks)`

Concatenation

Jika S_1 dan S_2 adalah string, maka concatenation dari S_1 dan S_2 yang dinotasikan sebagai $S_1//S_2$ adalah string yang mengandung karakter pada S_1 diikuti dengan karakter pada S_2.

Contoh:

a. Misalkan S_1 = 'MARK' dan S_2 = 'TWAIN' maka
   $S_1//S_2$ = 'MARKTWAIN', S_1//' '//S_2 = 'MARK TWAIN'

b. Concatenation dinyatakan dalam bahasa pemrograman:
   `Pascal  S1 + S1 atau CONCAT(S1, S2)`
   `Basic  S1 + S2`

Length

Jumlah karakter dalam suatu string disebut length (panjang) dan ditulis sebagai LENGTH(String).

Contoh:

a. LENGTH(‘COMPUTER’) = 8, LENGTH(‘’) = 0, LENGTH(‘ ‘) = 1

b. Length dinyatakan dalam bahasa pemrograman:

```
Pascal      LENGTH(String)
Basic       LEN(String)
```

4.5. Pengolahan Kata

Pada waktu dulu, data karakter yang diolah computer kebanyakan berkaitan dengan item data seperti nama dan alamat. Sekarang komputer juga digunakan untuk mengolah barang cetakan seperti surat, artikel dan laporan.

Operasi-operasi yang berkaitan dengan pengolahan kata adalah:

a. *Replacement*. Mengganti suatu string dalam teks dengan string lainnya.
b. *Insertion*. Menyisipkan suatu string ke dalam teks.
c. *Deletion*. Menghapus suatu string dari teks.

Operasi di atas dapat dijalankan menggunakan operasi string yang telah dibahas terdahulu.

Insertion

Misalkan kita ingin menyisipkan string S ke dalam teks T sehingga S berada pada posisi K. Operasi ini dinotasikan dengan INSERT(Teks, Posisi, String).

Contoh:

```
INSERT('ABCDEFG', 3, 'XYZ') = 'ABXYZCDEFG'
INSERT('ABCDEFG', 6, 'XYZ') = 'ABCDEXYZFG'
```

Fungsi INSERT ini dapat diimplementasikan menggunakan operasi string pada bagian yang lalu sebagai berikut:

INSERT(T, K, S) = SUBSTRING(T, 1, K – 1)//S//SUBSTRING(T, K, LENGTH(T) – K + 1)

Perhatikan, Substring awal T sebelum posisi K, yang mempunyai panjang K – 1, disambung dengan string S, dan hasilnya disambung dengan bagian sisa dari teks T, yang dimulai dari posisi K dan mempunyai panjang LENGTH(T) – (K – 1) = LENGTH(T) – K + 1.

Deletion

Misalkan kita ingin menghapus suatu substring mulai pada posisi K dengan panjang L dari teks T, maka operasi ini dinotasikan dengan DELETE(Teks, Posisi, Panjang).

Contoh:

```
DELETE('ABCDEFG', 4, 2) = 'ABCFG'
DELETE('ABCDEFG', 2, 4) = 'AFG'
```

Diasumsikan bahwa tidak ada yang dihapus jika K = 0. Maka
DELETE('ABCDEFG', 0, 2) = 'ABCDEFG'

Fungsi DELETE dapat diimplementasikan menggunakan operasi string sebagai berikut:

DELETE(T, K, L) = SUBSTRING(T, 1, K – 1)//SUBSTRING(T, K + L, LENGTH(T) – K – L + 1)

Perhatikan, substring awal dari T sebelum posisi K disambungkan dengan substring akhir dari T mulai dari posisi K + L. Panjang dari substring awal T adalah K – 1, dan panjang substring akhir adalah LENGTH(T) – (K + L – 1) = LENGTH(T) – K – L + 1.

Misalkan suatu teks T dan pola P, dan kita ingin menghapus pola P pertama yang ada pada teks T. Masalah ini dapat dipecahkan menggunakan fungsi DELETE, yaitu: DELETE(T, INDEX(T, P), LENGTH(P)).

Amati, pada teks T, pertama kita hitung INDEX(T, P), posisi dimana P muncul pertama kali, dan kemudian kita hitung LENGTH(P), jumlah karakter dalam P. Perhatikan bahwa bila INDEX(T, P) = 0 (artinya P tidak ada dalam T), teks T tidak dirubah.

Contoh:

a. Misalkan T = 'ABCDEFG' dan P = 'CD'. Maka INDEX(T, P) = 3 dan LENGTH(P) = 2. Karenanya DELETE('ABCDEFG', 3, 2) = 'ABEFG'.
b. Misalkan T = 'ABCDEFG' dan P = 'DC'. Kemudian INDEX(T, P) = 0 dan LENGTH(P) = 0. Karenanya DELETE('ABCDEFG', 0, 2) = 'ABCDEFG'.

Misalkan kita ingin menghapus semua pola P dalam text T, hal ini dapat dilakukan dengan melakukan DELETE(T, INDEX(T, P), LENGTH(P)) secara berulang-ulang sampai INDEX(T, P) = 0. Berikut adalah algoritma untuk memecahkan masalah tersebut.

```
Algoritma Suatu teks T dan pola P di memory.
          Algoritma ini menghapus semua pola P
          dalam T.

          1. [Mencari posisi P] Set K:= INDEX(T,
             P)
          2. Repeat While K ≠ 0:
             a. [Menghapus P dari T]
```

```
      Set T:= DELETE(T, INDEX(T, P),
LENGTH(P))
   b. [Update posisi P] Set K:=
      INDEX(T, P)
   [End of Loop]
3. Write: T
4. Exit.
```

Perlu ditekankan bahwa setelah setiap penghapusan, panjang T berkurang dan karenanya algoritma harus dihentikan. Bagaimanapun, jumlah perulangan yang dijalankan mungkin melebihi pemunculan P dalam teks T sebenarnya, perhatikan contoh di bawah ini:

Contoh:

a. Misalkan **Error! Reference source not found.** dijalankan menggunakan data T = ’XABYABZ’, P = ’AB’. Kemudian perulangan dalam algoritma akan dijalankan dua kali. Setelah penghapusan ’AB’ yang pertama maka T = ’XYABZ’. Seteleh penghapusan yang kedua, sisa ’AB’ di T dihapus maka nilai T = ’XYZ’. ’XYZ’ adalah hasilnya.
b. Misalkan **Error! Reference source not found.** dijalankan menggunakan data T = ’XAAABBBY’, P = ’AB’. Perhatikan bahwa pola ’AB’ hanya muncul satu kali pada teks T, akan tetapi perulangan dalam algoritma akan di jalankan tiga kali. Karena setelah penghapusan ’AB’ yang pertama maka T = ’XAABBY’, ternyata ’AB’ muncul lagi dalam teks T. Setelah ’AB’ dihapus kedua kalinya dari T, ’AB’ ternyata masih ada. Dan untuk terakhir kalinya ’AB’ dihapus maka nilai T:= ’XY’ dan ’AB’ tidak muncul lagi dalam T karena INDEX(T, P) = 0. Jadi ’XY’ adalah hasilnya.

Replacement

Misalkan kita ingin mengganti pola P_1 pertama dalam teks T dengan pola P_2, maka operasi ini dapat dinotasikan dengan REPLACE(Teks, $Pola_1$, $Pola_2$).

Contoh:

```
REPLACE('XABYABZ', 'AB', 'C') = 'XCYABZ'
REPLACE('XABYABZ', 'BA', 'C') = 'XABYABZ'
```

Pada kasus kedua, pola 'BA' tidak ada maka tidak terjadi perubahan.

Perhatikan bahwa fungsi REPLACE dapat diekspresikan sebagai suatu penghapusan disusul dengan penyisipan, jika kita menggunakan fungsi DELETE dan INSERT. Secara khusus, fungsi REPLACE dapat dijalankan menggunakan tiga langkah berikut:

```
K:= INDEX(T, P1)
T:= DELETE(T, K, LENGTH(P1))
INSERT(T, K, P2)
```

Dua langkah pertama adalah untuk menghapus P1 dari T, sedangkan langkah ketiga menyisipkan P2 diposisi K dimana P1 dihapus.

Misalkan terdapat teks T, pola P dan Q di memory computer. Misalkan kita ingin mengganti semua pola P yang terdapat dalam T dengan pola Q. Hal ini dapat dilaksanakan dengan menerapkan REPLACE(T, P, Q) berulang kali sampai INDEX(T, P) = 0. Perhatika algoritma di bawah ini:

```
Algoritma Suatu teks T, pola P dan Pola Q di
          memory. Algoritma ini mengganti semua
          pola P pada teks T dengan pola Q.

          1. [Cari Index P] Set K:= INDEX(T, P)
```

```
2. Repeat While K ≠ 0
   a. [Ganti P dengan Q] Set T:=
      REPLACE(T, P, Q)
   b. [Update Index] Set K:= INDEX(T,
      P)
   [End of Loop
3. Write: T
4. Exit.
```

Contoh:

a. Misalkan **Error! Reference source not found.** dijalankan dengan data T = ‘XABYABZ’, P = ‘AB’, Q = ‘C’. Kemudian perulangan dalam algoritma akan dijalankan dua kali. Pada perulangan pertama, ’AB’ dalam T digantikan dengan ’C’ sehingga T = ’XCYABZ’. Pada perulangan kedua, sisa ’AB’ dalam T diganti dengan ’C’ sehingga T = ’XCYCZ’.

b. Misalkan **Error! Reference source not found.** dijalankan dengan data T = ’XAY’, P = ’A’ dan Q = ’AB’. Perhatikan, algoritma tidak akan pernah berakhir. Alasannya adalah P akan selalu ada dalam teks T, tidak perduli berapa kali perulangan dijalankan.

T = ’XABY’ → perulangan pertama
T = ’XAB^{2}Y’ → perulangan kedua
...
T = ’XAB^{n}Y’

4.6. Algoritma Pattern Matching

Pattern Matching adalah masalah yang memutuskan apakah pola P ada dalam teks T atau tidak. Kita asumsikan bahwa panjang P tidak melebih panjang T. Bagian ini menjelaskan dua algoritma *pattern matching*. Kita juga akan membahas kompleksitas algoritma jadi kita bisa membandingkan efisiensinya.

Catatan: selama pembahasan algoritma pattern matching, karakter kadang-kadang dinotasikan menggunakan huruf kecil (a, b, c,...) dan perpangkatan digunakan untuk menotasikan pengulangan, seperti contoh: $a^2b^3ab^2$ untuk aabbbabb dan $(cd)^3$ untuk cdcdcd.

Sebagai tambahan, empty string akan dinotasikan menggunakan Λ (lambda), dan concatenation string X dan Y akan dinotasikan dengan X · Y atau XY.

Algoritma Pattern Matching Pertama

Algoritma pattern matching pertama dimana kita membandingkan pola P dengan tiap substring dalam T, bergerak dari kiri ke kanan, sampai kita temukan persamaan. Lebih detail, perhatiakn W_K = SUBSTRING(T, K, LENGTH(P)).

Misalkan W_K dinotasikan sebagai substring dari T dengan panjang yang sama dengan P dan dimulai pada karakter ke-K dari T. Pertama kita bandingkan P, karakter per karakter dengan substring pertama, W_1. Jika semua karakter sama, maka P = W_1 dan berarti P ada di T dan INDEX(T, P) = 1. Di lain sisi, misalkan kita menemukan bahwa beberapa karakter tidak sama dengan karakter W_1. Maka P ≠ W_1, dan kita dapat segera berpindah ke substring berikutnya, W_2. Berikutnya kita bandingkan P dengan W_2. Jika P ≠ W_2, kemudian kita bandingkan lagi P dengan W_3 dan seterusnya. Proses berhenti:

a. Jika kita menemukan persamaan antara P dengan beberapa substring W_K dan berarti P ada dalam T dan INDEX(T, P) = K, atau
b. Jika kita telah menghabiskan semua W_K tanpa ada persamaan dan berari P tidak ada dalam T dan INDEX(T, P) = 0.

Maksimum nilai MAX dari subscript K adalah LENGTH(T) – LENGTH(P) + 1.

Mari kita asumsikan seperti ilustrasi di atas. Jika P adalah string dengan 4 karakter dan T adalah string dengan 20 karakter, dan bahwa

T dan P ada di memory komputer sebagai suatu linear array dengan satu karakter per elemen. Berarti P = P[1]P[2]P[3]P[4] dan T = T[1]T[2]T[3]...T[19]T[20].

Kemudian P dibandingkan dengan 4 karakter substring dari T, dimana:
W1 = T[1]T[2]T[3]T[4], W2 = T[2]T[3]T[4]T[5],... W17 = T[17]T[18]T[19]T[20].
Terdapat 17 substring dalam T, yaitu MAX = 20 – 14 + 1 = 17.

Bentuk algoritma formal, dimana P adalah string dengan r karakter dan T adalah string dengan s karakter, dapat dilihat pada algoritma di bawah:

```
Algoritma (Pattern Matching) P dan T adalah
          string dengan panjang masing-masing R
          dan S, dan disimpan dalam array dengan
          satu karakter per elemen. Algoritma
          ini mencari nilai INDEX dari P dalam
          T.

          1. [Inisialisasi] Set K:= 1 dan
             MAX:= S - R + 1.
          2. Repeat Step 3 to 5 While K ≤
             MAX:
          3.    Repeat For L = 1 to R
             [Menguji tiap karakter dari P]
                   If P[L] ≠ T[K + L - 1],
          then Go To Step 5.
               [End of Inner Loop]
          4.    [Sukses] Set INDEX = K and
             Exit.
          5.    Set K:= K + 1.
             [End] of Step 2 Outer Loop]
```

```
6. [Gagal] Ste INDEX:= 0.
7. Exit.
```

Perhatikan bahwa algoritma di atas mengandung dua perulangan, satu di dalam lainnya. Perulangan bagian luar menjalankan tiap substring dengan R karakter W_K = T[K]T[K+1]...T[K+R-1] dari T. Perulangan bagian dalam membandingkan P dengan W_K, karakter per karakter. Jika ada karakter yang tidak sama maka kontrol berpindah ke Step 5, yang menambah nilai K dan maju ke substring berikutnya dari T. Jika semua karakter dari P sama dengan beberapa W_K, berarti P ada dalam T dan K adalah INDEX dari P dalam T. Sebaliknya, jika perulangan bagian luar telah menyelesaikan seluruh siklusnya, berarti P tidak ada dalam T dan INDEX = 0.

Kompleksitas dari algoritma pattern matching ini diukur berdasarkan jumlah C perbandingan antara karakter dalam pola P dan karakter dalam teks T. Untuk mencari nilai C, misalkan N_k kita notasikan sebagai jumlah perbandingan yang terjadi pada perulangan bagian dalam ketika P dibandingkan dengan WK. Maka C = N_1 + N_2 +... + N_L, dimana L adalah posisi L dalam T dimana P pertama kali ditemukan atau L = MAX jika P tidak ada dalam T. Contoh beriku ini menghitung C untuk beberapa P dan T dimana LENGTH(P) = 4 dan LENGTH(T) = 20 dan berarti MAX = 20 – 4 + 1 = 17.

Contoh:

a. Misalkan P = ’aaba’ dan T = ’cdcd...cd’ = $(cd)^{10}$. Jelas bahwa P tidak ada dalam T dan untuk tiap siklus dari 17 siklus, nilai N_k = 1, karena karakter pertama P tidak sama dengan substring W_K. Maka C = 1 + 1 +.... + 1 = 17.
b. Misalkan P = ’aaba’ dan T = ababaaba.... Perhatikan bahwa P adalah substring dari T. Faktanya P = W_5 dan N_5 = 4. Juga, bila kita bandingkan P dengan W_1 = ’abab’, kita dapat melihat bahwa N_1 = 2, karena karakter pertama sama tetapi karakter kedua berbeda, akan tetapi bila dibandingkan dengan W_2 = ’baba’, dapat dilihat bahwa N_2 = 1, karena karakter pertama

tidak cocok. Dengan cara yang sama, $N_3 = 2$ dan $N_4 = 1$. Maka $C = 2 + 1 + 2 + 1 + 4 = 10$.

c. Jika P = 'aaab' dan T = aa...a = a^{20}. P tidak terdapat dalam T. Dan setiap W_K = 'aaaa', maka $N_K = 4$, karena tiga karakter pertama P sama dengan W_K. Maka $C = 4 + 4 + ... + 4 = 17 * 4 = 68$.

Secara umum, jika P adalah string dengan r karakter dan T adalah string dengan s karakter, maka ukuran data untuk algoritma adalah $n = r + s$.

Kasus terburuk (*Worst case*) ada jika setiap karakter P kecuali karakter terakhir sama dengan setiap karakter W_K, seperti pada Contoh. Dalam hal ini, $C(n) = r (s – r + 1)$. Untuk jumlah n, kita temukan bahwa $s = n – r$, maka $C(n) = r(n – 2r + 1)$.

Maksimum nilai C(n) didapat bila $r = (n + 1)/4$. Maka jika nilai r disubtitusi pada formula di atas $C(n) = \frac{(n+1)^2}{8}$.

Kompleksitas Average Case dalam beberapa situasi tergantung pada probabilitas yang biasanya tidak diketahui. Jika karakter P dan T secara aca dipilih dari beberapa alfabet. Kompleksitas average case tetap tidak mudah untuk dianalisa, tetapi kompleksitas average case tetap suatu faktor dari *worst case*.

Algoritma Pattern Matching Kedua

Algoritma *pattern matching* kedua menggunakan suatu tabel yang diturunkan dari pola P tertentu tetapi bebas terhadap teks T. Misalkan P = 'aaba'. Pertama akan dijelaskan mengenai tabel dan cara penggunaannya. Mislkan $T = T_1T_2T_3...$, dimana T_i dinotasikan sebagai karakter ke-i dari T; dan jika dua karakter pertama dari T sama dengan P, misalkan T = 'aa..'. Maka T akan mempunyai salah satu bentuk di bawah ini:

a. T = 'aab..',
b. T = 'aaa..',
c. T = 'aax'

Dimana x adalah sembarang karakter yang berbeda dengan a atau b. Misalkan kita baca T_3 dan ditemukan T_3 = 'b'. Kemudian kita baca berikutnya T_4 untuk melihat apakah T_4 = 'a', yang akan memberikan kesamaan antara P dengan W_1. Dilain sisi, misalkan T_3 = 'a'. Maka kita tahu bahwa $P \neq W_1$; tetapi kita juga tahu bahwa W_2 = 'aa..', sama dengan dua karakter pertama P. Karena itu karakter berikutnya yang kita baca adalah T_4 untuk mengetahui apakah T_4 = 'b'. Terakhir, misalkan T_3 = x. Maka kita tahu bahwa $P \neq W_1$, $P \neq W_2$ dan $P \neq W_3$, karena x tidak terdapat pada P. Karena itu berikutnya kita baca T_4 untuk melihat apakah T_4 = 'a', untuk melihat apakah karakter pertama dari W_4 sama dengan karakter pertama pada P.

Terdapat dua hal penting pada prosedur di atas. Pertama, jika kita baca T_3 kita hanya perlu membandingkan T_3 dengan karakter yang ada pada P. Jika tidak ada yang sama, berarti kita berada pada kasus terakhir, dimana karakter x tidak terdapat pada P. Kedua, setelah membaca dan memeriksa T_3, berikutnya yang kita baca adalah T_4; kita tidak perlu kembali lagi ke teks T. Gambar di bawah menunjukkan tabel yang digunakan dalam algoritma pattern matching yang kedua untuk pola P = 'aaba'. Pada tabel dan graph, pola P dan substringnya Q diwakili oleh huruf besar miring. Tabel diperoleh dengan cara sebagai berikut. Pertama, kita notasikan Q_i sebagai substring awal dari P dengan panjang i, maka

$$Q_0 = \Lambda,\ Q_1 = a,\ Q_2 = a^2,\ Q_3 = a^2b,\ Q_4 = a^2ba = P.$$

Baris dari tabel diberi label substring awal tersebut. Kolom diberi label a, b dan x, dimana x mewakili sembarang karakter yang tidak ada

dalam pola P. Misalkan f adalah fungsi yang ditentukan oleh tabel, f(Q_1, t) dinotasikan sebagai masukan dalam tabel pada baris Q_1 dan kolom t (dimana t adalah sembarang karakter). Masukan f(Q_1, t) ini didefinisikan sebagai Q terbesar yang ada sebagai substring akhir dari string Q_it, gabungan antara Q_i dan t. Sebagai contoh:

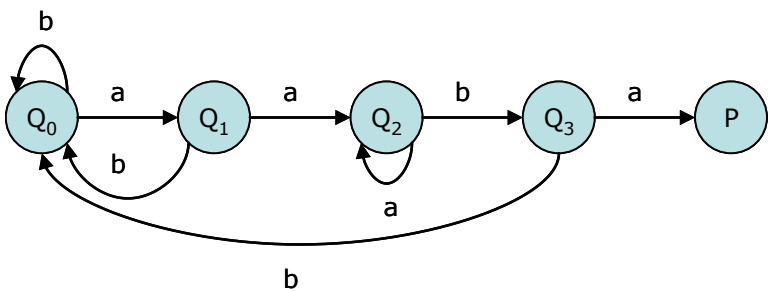
Q_0a = a, f(Q_0, a) = Q_1. Karena a = Q_1 adalah substring akhir terbesar.
Q_0b = b, f(Q_0, b) = Q_0. Karena Tidak ada substring akhir terbesar maka Q_0.
Q_1a = aa = a^2, f(Q_1, a) = Q_2. Karena a^2 = Q_2 adalah substring akhir terbesar.
Q_1b = ab, f(Q_1, b) = Q_0. Karena Tidak ada substring akhir terbesar maka Q_0.
Q_2a = a^2a = a^3, f(Q_2, a) = Q_2. Karena a = Q_1 dan a^2 = Q_2 adalah substring akhir dari Q_it. Tapi Q_2 adalah substring akhir terbesar.
Q_2b = a^2b, f(Q_2, a) = Q_3. Karena a^2b = Q_3 adalah substring akhir terbesar.
Q_3a = a^2ba, f(Q_3, a) = P. Karena a^2ba = P adalah substring akhir terbesar.
Q_3b = a^2bb, f(Q_3, b) = Q_0. Karena Tidak ada substring akhir terbesar maka Q_0.

Perlu diingat bahwa f(Q_i, x) = Q_0 untuk semua Q, karena x tidak ada dalam pola P. Karena itu kadang-kadang kolom x biasanya dihilangkan dari tabel.

	a	b	x
Q_0	Q_1	Q_0	Q_0
Q_1	Q_2	Q_0	Q_0
Q_2	Q_2	Q_3	Q_0
Q_3	P	Q_0	Q_0

Gambar 4. 9. Algoritma Q

Tabel pada gambar di atas juga dapat digambarkan menggunakan graph seperti pada gambar dibawah Graph tersebut didapatkan dengan cara: pertama gambar setiap node yang mewakili setiap substring awal dari Q_i dari P. Q disebut dengan states (keadaan) dari sistem, dan Q_0 dinyatakan sebagai keadaan awal (initial states). Kedua, tarik garis panah sesuai dengan tiap masukan pada tabel. Secara khusus jika $f(Q_i, t) = Q_j$, maka ada garis panah yang diberi label karakter t dari Q_i ke Q_j. Sebagai contoh, $f(Q_2, b) = Q_3$, jadi ada garis panah berlabel b dari Q_2 ke Q_3.



Gambar 4. 10. Algoritma Q

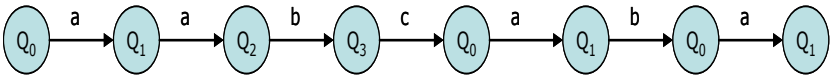
Misalkan $T = T_1T_2T_3... T_N$ dinotasikan sebagai string dengan n karakter yang akan dicari menggunakan pola P = 'aaba'. Dimulai dengan keadaan awal Q_0 dan menggunakan teks T, kita akan mendapatkan urutan dari keadaan S_1, S_2, S_3,... dan seterusnya. Misalkan $S_1 = Q_0$ dan kita membaca karakter pertama dari T. Kemudian dari tabel

dan graph pada **Error! Reference source not found.** dan **Error! Reference source not found.**, dari pasangan (S_1, T_1) didapat keadaan kedua yaitu F(S_1, T_1) = S_2. Kita baca karakter berikutnya yaitu T_2. Pasangan (S_2, T_2) menyatakan keadaan ketiga yaitu F(S_2, T_2) = S_3 dan seterusnya. Ada dua kemungkinan:

a. Ada keadaan S_K = P, pola yang diinginkan. Dalam hal ini P ada dalam T dan indexnya adalah K – LENGTH(P).
b. Tidak ada keadaan dari S_1, S_2,... S_{N+1} yang sama dengan P. Dalam hal ini P tidak terdapat dalam T.

Contoh:

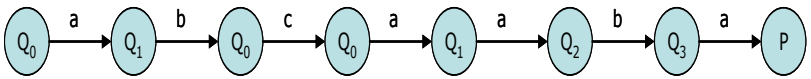
a. Misalkan T = ‘aabcaba’. Dimulai dari Q_0, kita menggunakan karakter T dan graph atau table kita akan mendapatkan urutan keadaaan sebagai berikut:



Gambar 4. 11. Algoritma Q

Kita tidak akan menemukan keadaan P, jadi P tidak ada dalam T.

b. Misalkan T = ’abcaabaca’. Maka kita akan mendapatkan urutan keadaan sebagai berikut:



Gambar 4. 12. Algoritma Q

Disini kita akan menemukan pola P sebagai keadaan S_8. Berarti P ada dalam T dan indeksnya adalah 8 – LENGTH(P) = 4.

Algoritma (Pattern Matching). Tabel pattern matching F(Q_1, T) dari suatu pola P ada dalam memory, dan input adalah String

T dengan N karakter, T = $T_1T_2...T_N$. Algoritma ini akan mencari INDEX P dalam teks T.

1. [Inisialisasi] Set K:= 1 and S_1 = 'Q0'.
2. Repeat Step 3 to 5 While $S_K \neq$ 'P' and K ≤ N
3. Read T_K.
4. Set S_{K+1}:= F(S_K, T_K).
5. Set K:= K + 1.

   [End of Step 2 Loop]
6. [Sukses]

   If S_K = P, then:

   INDEX:= K – LENGTH(P)

   Else:

   INDEX:= 0.

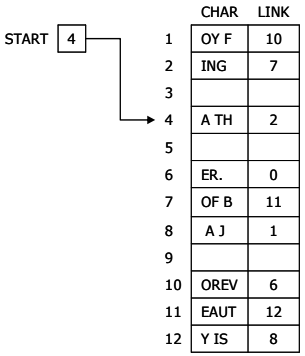
   [End of If Structure]
7. Exit.

Waktu yang dibutuhkan untuk menjalankan algoritma di atas proporsional terhadap jumlah perulangan pada Step 2. *Worst case* ada apabila seluruh teks T dibaca, dalam hal ini perulangan dijalankan sebanyak n = LENGTH(T) kali.

4.7. Soal Latihan

1. Misalkan W adalah string ABCD.
   a. Tentukan panjang W.
   b. Buat daftar semua substring dari W.
   c. Buat daftar semua initial substring dari W
2. Berikan deskripsi singkat dari tiga tipe struktur yang digunakan untuk menyimpan string.

3. Tentukan string yang tersimpan pada gambar dibawah, diasumsikan link dengan nilai 0 adalah akhir dari list.



	CHAR	LINK
1	OY F	10
2	ING	7
3		
4	A TH	2
5		
6	ER.	0
7	OF B	11
8	A J	1
9		
10	OREV	6
11	EAUT	12
12	Y IS	8

4. Berikan beberapa keuntungan dan kerugian penggunaan linked storage untuk menyimpan string.
5. Jelaskan dengan singkat:
   a. Statik
   b. Semistatik
   c. Variabel karakter dinamis.

BAB V

PERCABANGAN DAN PERULANGAN

Capaian Pembelajaran:

1. Mampu menjelaskan percabangan dan perulangan
2. Mampu memahami teknik percabangan
3. Mampu memahami teknik perulangan

5.1. Teknik Percabangan

Agar lebih memahami apa itu percabangan, kita dapat merujuk pada beberapa teknik percabangan berikut ini:

IF-tunggal

Digunakan pada saat kita dihadapkan pada satu kasus saja. Misalnya menentukan bilangan genap.

Bentuk umum penulisan:

IF (kondisi) THEN pernyataan;

Contoh:

IF x mod 2 = 0 THEN writeln (x, ‘ bilangan genap’);

Multiple Condition

Jika kita dihadapkan dengan dua kasus dapat digunakan IF-Then-Else. Misalnya menentukan bilangan genap atau ganjil.

Bentuk umum penulisan:

```
IF (kondisi) THEN
        Pernyataan1
ELSE
        Pernyataan2;
```

Pada saat kondisi terpenuhi (bernilai True), maka akan dilakukan pernyataan 1. Tetapi jika kondisi tidak terpenuhi (bernilai False), maka akan melakukan pernyataan dibawah ELSE yaitu pernyataan 2.

Contoh:

```
IF x mod 2 = 0 THEN
        writeln (x, ' bilangan genap')
ELSE
        writeln (x, ' bilangan ganjil');
```

Jika kita dihadapkan pada kondisi dengan 3 kasus atau lebih, dapat digunakan IF-Majemuk. Misalnya menentukan apakah sebuah bilangan termasuk bilangan positif, negatif atau nol.

Bentuk umum penulisan:

```
IF (kondisi 1) THEN
        Pernyataan1
ELSE
        IF (kondisi 2) THEN
                Pernyataan2
        ELSE
                IF (kondisi 3) THEN
                        Pernyataan3
                ELSE
                        Pernyataan4;
```

Yang perlu diperhatikan dalam bahasa pemrograman Pascal, pernyataan diatas ELSE tidak boleh diakhiri dengan tanda titik koma.

Contoh:

```
IF x <0 THEN
        writeln (x, ' bilangan negatif')
ELSE
        IF x >0 THEN
                writeln (x, ' bilangan positif')
        ELSE
                writeln (x, ' adalah nol');
```

Case

Bentuk penulisan IF-Majemuk dapat disederhanakan menggunakan perintah CASE. Dengan CASE, percabangan disajikan seperti pemilihan menu sehingga bentuk penulisannya lebih mudah dan sederhana.

Bentuk umum penulisan:

```
CASE ekspresi OF
        Nilai 1: pernyataan 1;
        Nilai 2: pernyataan 2;
        Nilai 3: pernyataan 3;
        .
        .
        .
        Nilai n: pernyataan n;
ELSE
        Pernyataan x;
END;
```

Contoh: mencetak nama-nama hari dalam seminggu.

```
CASE nomor_hari OF
        1: Writeln (‘Senin’);
        2: Writeln (‘Selasa’);
        3: Writeln (‘Rabu’);
        4: Writeln (‘Kamis’);
        5: Writeln (‘Jumat’);
        6: Writeln (‘Sabtu’);
        7: Writeln (‘Minggu’);
ELSE
        Writeln (‘Angka yang anda masukkan salah!’);
END;
```

IF-Bersarang

Pada permasalah percabangan dengan kondisi bertingkat, kita dapat menggunakan If-bersarang (nested-IF). Secara umum, if bersarang adalah if yang didalamnya terdapat if. Sehingga kondisinya bertingkat yang harus dipenuhi untuk melakukan sebuah pernyataan.

Bentuk umum penulisan:

```
IF (kondisi 1) THEN
        IF (kondisi 2) THEN
                IF (kondisi 3) THEN
                        Pernyataan1
                ELSE
                        Pernyataan2
        ELSE
                Pernyataan3
ELSE
Pernyataan4
```

5.2. Teknik Perulangan

Agar lebih memahami apa itu percabangan, kita dapat merujuk pada beberapa teknik percabangan berikut ini:

Berbagai Statemen Perulangan

FOR-menaik

Bentuk umum penulisan:

```
FOR pencacah:= awal TO akhir DO
        Pernyataan;
```

FOR-menurun

Bentuk umum penulisan:

```
FOR pencacah:= awal DOWNTO akhir DO
        Pernyataan;
```

WHILE-DO

Pernyataan akan terus diulang selama kondisi terpenuhi (bernilai TRUE). Bentuk umum penulisan:

```
        Pencacah:= nilai;
```

```
WHILE kondisi DO
Begin
        Pernyataan;
        Pencacah:= Pencacah + 1;
End;
```

REPEAT-Until

Pernyataan akan terus diulang sampai kondisi terpenuhi (bernilai TRUE). Bentuk umum penulisan:

```
Pencacah:= nilai;
REPEAT
        Pernyataan;
        Pencacah:= Pencacah + 1;
UNTIL kondisi;
```

Perulangan Bersarang

Perulangan bersarang yaitu suatu perulangan proses yang di dalamnya terdapat perulangan proses yang lain. Kegunaannya untuk pengolahan data yang banyak dan setiap data memiliki sejumlah sub-data lainnya. Contoh: sejumlah mahasiswa mengambil sejumlah mata kuliah yang berbeda-beda.

Bentuk umum penulisan:

a. FOR bersarang

```
FOR pencacah_1:= awal TO/DOWNTO akhir DO
Begin
        FOR pencacah_2:= awal TO/DOWNTO akhir
DO
        Pernyataan;
End;
```

b. While bersarang

```
Pencacah_1:= nilai;
WHILE kondisi_1 DO
Begin
```

```
                    Pencacah_2:= nilai;
                    WHILE kondisi_2 DO
                    Begin
                        Pernyataan;
                        Pencacah_2:=  Pencacah_2  +
                    1;
                    End;
                    Pernyataan;
                    Pencacah_1:= Pencacah_1 + 1;
            End;
```

c. Repeat bersarang

```
            Pencacah_1:= nilai;
            REPEAT
                    Pernyataan;
                    Pencacah_2:= nilai;
                    REPEAT
                        Pernyataan;
                        Pencacah_2:=  Pencacah_2  +
                    1;
                    UNTIL kondisi_2;
                    Pencacah_1:= Pencacah_1 + 1;
UNTIL kondisi_1;
```

5.3. Soal Latihan

1. Sebutkan dan jelaskan teknik percabangan!
2. Sebutkan dan jelaskan teknik perulangan!

BAB VI

ARRAY, RECORD DAN POINTER

Capaian Pembelajaran:

1. Mampu menjelaskan tentang penggunaan array.
2. Mampu menjelaskan tentang penggunaan record.
3. Mampu menjelaskan tentang penggunaan pointer

6.1. Array

Array linear adalah daftar dengan n jumlah data yang homogen (data dengan tipe data yang sama), yang berarti:

a. Elemen dalam array direferensikan menggunakan kumpulan index yang teridiri dari n angka yang berurutan.
b. Elemen dalam array disimpan disimpan pada lokasi memory yang berurutan.

Jumlah eleman suatu array disebut dengan panjang atau ukuran array. Jika tidak dinyatakan lain, kita asumsikan kumpulan indeks terdiri dari angka integer 1, 2,..., n. Secara umum, panjang atau jumlah elemen data dalam array diperoleh menggunakan rumus Length = UB – LB + 1, dimana UB (Upper Bound) adalah indeks terbesar atau biasa disebut batas atas, dan LB (Lower Bound) adalah indeks terkecil atau disebut dengan batas bawah. Length = UB jika LB = 1.

Elemen suatu array A dinotasikan menggunakan subscript A1, A2, A3,..., An atau menggunakan notasi kurung (digunakan dalam bahasa BASIC) A(1), A(2), …, A(N) atau menggunakan notasi kurung siku (digunakan dalam bahasa PASCAL) A[1], A[2], …, A[N].

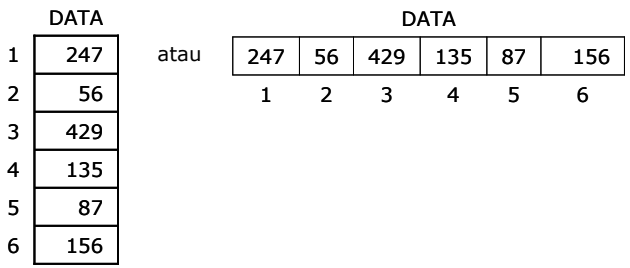
Kita akan menggunakan subscript atau notasi kurung siku. Berdasarkan notasi di atas, angka K dalam A[K] disebut subscript atau indeks dan A[K] disebut subscripted variable.

Contoh:

a. Misalkan DATA adalah array linear dengan 6 elemen data integer, dengan data sebagai berikut: DATA[1] = 247, DATA[2] = 56, DATA[3] = 429, DATA[4] = 135, DATA[5] = 87 dan DATA[6] = 156.

   Kadang-kadang kita menotasikan array menggunakan: DATA: 247, 56, 429, 135, 87, 156.

   Array DATA biasanya digambarkan seperti pada gambar di bawah:



Gambar 6. 1. Array Data

b. Suatu perusahaan mobil menggunakan array AUTO untuk merekam jumlah penjualan mobil setiap tahun mulai tahun 1932 sampai 1984. Daripada menggunakan index mulai dari 1, lebih berguna jika index dimulai dengan 1932 sehingga AUTO[K] = jumlah mobil yang terjual pada tahun K. Maka LB = 1932 adalah batas bawah dan UB = 1984 adalah batas atas dari AUTO. Length = UB – LB + 1 = 1984 – 1932 + 1 = 55. Berarti AUTO mempunyai 55 elemen dan kumpulan indeksnya adalah integer mulai 1932 sampai 1984.

Tiap bahasa pemrograman mempunyai aturan sendiri dalam mendeklarasikan array. Tiap deklarasi setidaknya mempunyai 3 item informasi:

a. Nama array
b. Tipe data eleman array
c. Kumpulan indeks array.

Contoh:

a. Misalkan DATA adalah array linear dengan 6 elemen bilangan real. Berikut adalah deklarasi variable DATA pada beberapa bahasa pemrograman:

FORTRAN:	REAL DATA(6)
PL/1:	DECLARE DATA(6) FLOAT
Pascal :	VAR DATA: ARRAY[1..6] OF REAL;

b. Perhatikan array integer AUTO dengan batas bawah LB = 1932 dan batas atas UB = 1984. Berikut ini deklarasi variabel AUTO pada beberapa bahasa pemrograman:

FORTRAN:	INTEGER AUTO(1932:1984)
PL/1:	DECLARE AUTO(1932:1984) FIXED
PASCAL:	VAR AUTO: ARRAY[1932..1984] of Integer;

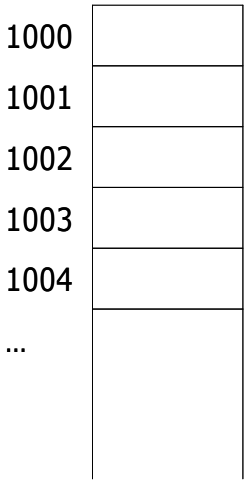
Kita akan menotasikan array dengan menulis AUTO(1932:1984) .

Beberapa bahasa pemrograman (Fortran dan Pascal) mengalokasikan ruang memory untuk array secara statis, selama kompilasi program; karena itu ukuran array tetap selama program dijalankan. Di lain sisi, beberapa bahasa pemrograman membolehkan kita untuk membaca suatu integer n dan kemudian mendeklarasikan suatu array dengan n elemen; bahasa pemrograman ini mengalokasikan memori secara dinamis.

Representasi Array Linear di Memori

Misalkan LA adalah suatu array linear dalam memori komputer. Ingat bahwa memori komputer secara sederhana adalah urutan lokasi alamat seperti pada gambar di bawah. Misalkan kita menggunakan notasi LOC(LA[K]) = alamat elemen LA[K] dari array LA.

Seperti dinyatakan sebelumnya, elemen LA disimpan pada sel memori secara berurutan. Maka, komputer tidak perlu mengingat setiap alamat dari elemen LA, tetapi hanya menyimpan alamat pertama dari elemen LA, yang dinotasikan dengan BASE(LA) dan disebut dengan alamat awal LA. Menggunakan alamat BASE(LA), komputer menghitung alamat tiap elemen LA menggunakan rumus: LOC(LA[K]) = BASE(LA) + q(K – Lower Bound), dimana w adalah jumlah word per sel memory untuk array LA. Perhatikan bahwa waktu untuk menghitung LOC(LA[K]) secara esensial sama untuk semua nilai K. Lebih jauh, untuk sembarang K, seseorang dapat menempatkan dan mengakses isi dari LA[K] tanpa membaca elemen lain dari LA.



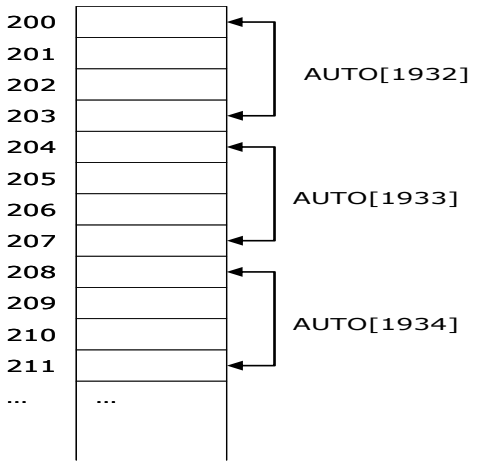
Gambar 6. 2. Elemen LA

Contoh:

Perhatikan array AUTO pada Contohb, yang merekam penjualan mobil tiap tahun mulai tahun 1932 sampai 1984. Misalkan AUTO ada dalam memori computer seperti pada gambar di bawah. Diketahui, Base(AUTO) = 200 dan w = 4 words per sel memori untuk AUTO. Maka LOC(AUTO[1932]) = 200, LOC(AUTO[1933]) = 204, LOC(AUTO[1934]) = 208, …

Alamat elemen array untuk tahun K = 1965 dapat dihitung menggunakan persamaan LOC(LA[K]) = BASE(LA) + q(K – Lower Bound):

LOC(AUTO[1965]) = BASE(AUTO) + w(1965 – 1932) = 200 + 4(1965 – 1932) = 332.



Gambar 6. 3. LOC

Menjelajahi Linear Array

Misalkan A adalah koleksi elemen data yang disimpan dalam memory computer. Misalkan kita ingin mencetak isi dari tiap elemen A atau misalkan kita ingin menghitung jumlah elemen A. Hal ini dapat dilakukan dengan menjelajahi A, dengan mengakses dan memproses tiap elemen A tepat satu kali.

Algoritma berikut menjelajahi linear array LA. Kesederhanaan algoritma ini karena LA adalah struktur linear. Struktur linear lainnya, seperti linked list, dapat juga dengan mudah dijelajahi. Di lain sisi, menjelajahi struktur non-linear, seperti tree dan graphs akan menjadi lebih rumit.

```
Algoritma (Menjelajahi Linear Array) LA adalah
          suatu linear array dengan batas bawah
          LB dan batas atas UB. Algoritma ini
          menjelajahi LA dan melakukan operasi
          PROCESS untuk tiap elemen LA.

          1. [Inisialisasi Counter] Set K:= LB.
          2. Repeat Step 3 dan 4 While K ≤ UB
          3.    [Mengunjungi    elemen]    Apply
             PROCESS to LA[K]
          4.    [Menambah Counter] Set K:= K + 1
             [End of Step 2 loop]
          5. Exit.
```

Algoritma berikut adalah bentuk lain dari

```
Algoritma (Menjelajahi Linear Array) Algoritma
          ini menjelajahi linear array LA dengan
          batas bawah LB dan batas atas UB.

        1. Repeat for K = LB to UB:
              Apply PROCESS to LA[K].
           [End of Loop]
        2. Exit.
```

Contoh:

Perhatikan array AUTO pada Contohb yang merekam jumlah penjualan mobil tiap tahun mulai tahun 1932 sampai 1984. Tiap modul berikut ini, melakukan operasi tertentu termasuk menjelajahi AUTO.

a. Hitung berapa NUM tahun yang mempunyai penjualan di atas 300 mobil.
    1. [Inisialisasi] Set NUM:= 0.
    2. Repeat For K = 1932 to 1984:
        If AUTO[K] > 300, then Set NUM:= NUM + 1.

[End of Loop].

3. Return

b. Cetak tahun dan jumlah mobil yang terjual pada tahun tersebut.

1. Repeat for K = 1932 to 1984:
   Write: K, AUTO[K].
   [End of loop]
2. Return.

Penyisipan dan Penghapusan

Misalkan A adalah kumpulan elemen data dalam memori computer. Inserting mengacu pada operasi menambahkan elemen lain ke dalam kumpulan A, dan Deleting mengacu pada operasi menghapus satu dari elemen A.

Menyisipkan suatu elemen di akhir suatu linear array dapat dengan mudah dilakukan dengan mengalokasikan ruang memory untuk array sehingga cukup besar untuk mengakomodasi penambahan elemen. Di lain sisi, misalkan kita perlu menyisipkan elemen di bagian tengah array. Maka, setidaknya setengah dari elemen-elemen harus digeser ke bawah ke lokasi baru untuk mengakomodasi elemen baru dan menjaga urutan elemen lainnya.

Dengan cara yang sama, menghapus elemen pada bagian akhir array adalah hal yang mudah, tetapi menghapus elemen di suatu tempat di bagian tengah array akan memerlukan pemindahan elemen yang berikutnya ke lokasi di atasnya untuk mengisi ruang yang kosong setelah penghapusan.

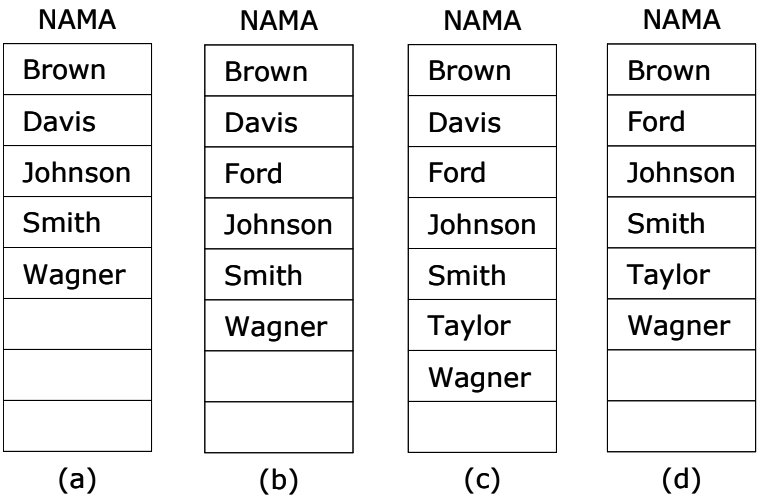
Contoh:

Misalkan array TEST dideklarasikan dengan 5 elemen data, tetapi data yang direkam hanya TEST[1], TEST[2], dan TEST[3]. Jika X adalah nilai test berikutnya, maka untuk menambahkan X ke dalam TEST dapat dinotasikan dengan TEST[4]:= X. Dengan cara yang sama,

jika Y adalah nilai test berikutnya, untuk menambahkan Y ke dalam TEST dapat dilakukan dengan TEST[5]:= Y. Sekarang, bagaimanapun, kita tidak bisa menambahkan nilai test baru ke dalam list.

Contoh:

Misalkan NAMA adalah array linear dengan 8 elemen, dan terdapat lima nama dalam array seperti pada gambar a di bawah. Perhatikan, nama disusun secara alfabet, dan kita ingin hal ini dipertahankan. Misalkan Ford ditambahkan ke dalam array. Maka Johnson, Smith dan Wagner harus digeser ke bawah sebanyak satu lokasi, lihat gambar b di bawah. Berikutnya, misalkan Taylor ditambahkan ke dalam array, maka Wagner harus digeser, lihat gambar c di bawah. Terakhir, misalkan Davis di hapus dari array. Maka lima nama yaitu Ford, Johnson, Smith, Taylor dan Wagner harus digeser naik sebanyak satu lokasi, lihat gambar d di bawah. Jelas bahwa pemindahan data akan menjadi sangat mahal jika terdapat ribuan nama dalam array.



Gambar 6. 4. Array Nama

Algoritma berikut menyisipkan suatu elemen data ITEM ke posisi ke-K dalam array linear LA dengan N elemen. Empat langkah pertama

menciptakan ruang dalam LA dengan memindahkan elemen mulai posisi ke-K sebanyak satu lokasi. Perlu ditekankan bahwa elemen dipindahkan dalam urutan mundur, misalkan, pertama LA[N], kemudian LA[N-1]... dan terakhir LA[K], selain itu data akan dihapus. Lebih detail, pertama set J:= N dan kemudian gunakan J sebagai counter, kurangi J tiap kali perulangan sampai J = K. Langkah berikutnya, Step 5, sisipkan ITEM ke dalam array pada ruang yang baru saja dibuat. Sebelum keluar dari algoritma, jumlah elemen N dalam LA ditambah 1 karena ada penambahan satu elemen baru.

```
Algoritma (Menyisipkan ITEM ke dalam array
          linear) INSERT(LA, N, K, ITEM)
          LA adalah array linear dengan N elemen
          dan K adalah bilangan integer positif
          dimana K ≤ N. Algoritma ini menyisipkan
          elemen ITEM pada posisi K dalam array
          LA.

          1. [Inisialisasi] Set J:= N.
          2. Repeat Step 3 and 4 while J ≥ K.
          3.    [Pindahkan elemen ke-J ke bawah]
            Set LA[J + 1]:= LA[J].
          4.    [Mengurangi Counter] Set J:= J -
            1.
            [End of Step 2 loop]
          5. [Menyisipkan elemen] Set LA[K]:=
            ITEM.
          6. [Reset N] Set N:= N + 1.
          7. Exit.
```

Algoritma berikut ini menghapus elemen ke-K dari array linear LA dan memasukkannya ke dalam variable ITEM.

```
Algoritma (Menghapus dari array linear)
          DELETE(LA, N, K, ITEM).
          LA adalah array linear dengan N elemen
          dan K adalah bilangan integer positif
          dimana K ≤ N. Algoritma ini menghapus
          elemen ke-K dari LA.

          1. Set ITEM:= LA[K].
          2. Repeat For J = K to N – 1:
                [Pindahkan elemen ke-[J+1] ke
          atas] Set LA[J]:= LA[J+1].
             [End of Loop]
          3. [Reset jumlah elemen dalam array]
             Set N:= N – 1.
          4. Exit.
```

Catatan: Perlu ditekankan bahwa jika terjadi banyak penghapusan dan penyisipan dalam koleksi elemen data, maka array linear mungkin bukan cara yang paling efisien untuk menyimpan data.

Pengurutan, Bubble Sort

Misalkan A adalah daftar dengan n jumlah data. Pengurutan A adalah operasi menyusun kembali elemen-elemen A sehingga A terurut dari kecil ke besar, sehingga A[1]<A[2]<A[3]<..<A[N].

Sebagai contoh, misalkan suatu list A berisi data 8, 4, 19, 2, 7, 13, 5, 16. Setelah diurutkan A menjadi 2, 4, 5, 7, 8, 13, 16, 19.

Pengurutan sepertinya merupakan hal yang sepele. Sebenarnya, pengurutan secara efisien mungkin menjadi hal yang cukup rumit. Terdapat banyak algoritma pengurutan yang berbeda. Berikut akan dibahas algoritma pengurutan yang paling sederhana yang dikenal dengan Bubble Sort.

Bubble Sort

Misalkan suatu daftar angka A[1], A[2],..., A[N] ada dalam memori. Algoritma bubble sort akan bekerja sebagai berikut:

Step 1. Bandingkan A[1] dan A[2] dan susun menurut urutan yang diinginkan, sehingga A[1] < A[2]. Kemudian bandingkan A[2] dan A[3] dan atur sehingga A[2] < A[3]. Lanjutkan sampai kita bandingkan A[N-1] dengan A[N] dan atur sehingga A[N-1] < A[N].

Perhatikan bahwa Step 1 melibatkan n – 1 perbandingan. (Selama Step 1, elemen terbesar diapungkan [bubble up] ke posisi ke-n atau ditenggelamkan (sink) ke posisi ke-n) Ketika Step 1 selesai, A[N] akan berisi elemen terbesar.

Step 2. Ulangi Step 1 dengan mengurangi satu perbandingan, perbandingan berhenti setelah kita membandingkan dan mengatur A[N-2] dan A[N-1]. (Step 2 melibatkan N – 2 perbandingan dan setelah Step 2 selesai, elemen terbesar kedua akan menempati A[N-1].

Step 3. Ulangi Step 1 dengan mengurangi dua perbandingan, perbandingan berhenti setelah kita membandingkan dan mengatur A[N-3] dan A[N-2].

...

Step N-1 Bandingkan A[1] dengan A[2] dan atur sehingga A[1] < A[2].

Setelah n – 1 langkah, daftar sudah terurut dari kecil ke besar.

Proses menjelajahi secara berurutan melalui seluruh bagian daftar biasanya disebut pass, jadi tiap langkah di atas disebut satu pass.

Berarti, algoritma bubble sort memerlukan n – 1 pass, dimana n adalah jumlah input.

Contoh:
Misalkan angka-angka berikut ini disimpan dalam array A:

32, 51, 27, 85, 66, 23, 13, 57

Kita akan menerapkan algoritma bubble sort pada array A.

Pass 1. Terdapat perbandingan sebagai berikut:

a. Bandingkan A1 dan A2, karena 32 < 51 maka tidak terjadi perubahan.
b. Bandingkan A2 dan A3, karena 51 > 27 maka letaknya ditukar menjadi ;
   32, **27**, **51**, 85, 66, 23,13, 57
c. Bandingkan A3 dan A4, karena 51 < 85 maka tidak terjadi pertukaran.
d. Bandingkan A4 dan A5, karena 85 > 66 maka terjadi pertukaran menjadi:
   32, 27, 51, **66**, **85**, 23,13, 57
e. Bandingkan A5 dan A6, karena 85 > 23 maka terjadi pertukaran menjadi:
   32, 27, 51, 66, **23**, **85**,13, 57
f. Bandingkan A6 dan A7, karena 85 > 13 maka terjadi pertukaran menjadi:
   32, 27, 51, 66, 23, **13**, **85**, 57
g. Bandingkan A7 dan A8, karena 85 > 57 maka terjadi pertukaran menjadi:
   32, 27, 51, 66, 23, 13, **57**, **85**

Pass 2. **27**, **32**, 51, 66, 23, 13, 57, 85
27, 32, 51, **23**, **66**, 13, 57, 85

27, 32, 51, 23, **13**, **66**, 57, 85
27, 32, 51, 23, 13, **57**, **66**, 85

Pass 3. 27, 32, **23**, **51**, 13, 57, 66, 85
27, 32, 23, **13**, **51**, 57, 66, 85

Pass 4. 27, **23**, **32**, 13, 51, 57, 66, 85
27, 23, **13**, **32**, 51, 57, 66, 85

Pass 5. **23**, **27**, 13, 32, 51, 57, 66, 85
23, **13**, **27**, 32, 51, 57, 66, 85

Pass 6. **13**, **23**, 27, 32, 51, 57, 66, 85

Pass 7. Terakhir, A1 dibandingkan dengan A. Karena 13 < 23 maka tidak terjadi pertukaran.

```
Algoritma (Bubble Sort) BUBBLE(DATA, N)
  DATA adalah array dengan N elemen. Algoritma
  akan mengurutkan elemen dalam DATA.

        1. Repeat Step 2 and 3 For K = 1 to N
           - 1
        2.    Set PTR:= 1.
        3.    Repeat While PTR ≤ N - K
              a. If DATA[PTR] > DATA[PTR + 1],
                 then:
                    Interchange DATA[PTR] and
              DATA[PTR + 1]
                 [End of Structure]
              b. Set PTR:= PTR + 1.
              [End of Inner Loop]
           [End of Outer Loop]
        4. Exit.
```

Perhatikan pada perulangan bagian dalam (inner loop) dikontrol oleh variable PTR, dan perulangan bagian luar (outer loop)

dikendalikan oleh indeks K. Perhatikan juga bahwa PTR juga digunakan sebagai subscript sedangkan K hanya sebagai counter.

Kompleksitas Algoritma Bubble Sort

Secara tradisional, waktu untuk melakukan algoritma pengurutan diukur berdasarkan jumlah perbandingan. Jumlah f(n) perbandingan dalam bubble sort dapat dengan mudah dihitung. Secara khusus, ada n – 1 perbandingan pada putaran pertama, yang menempatkan elemen terbesar pada posisi akhir. Terdapat n – 2 perbandingan pada putaran kedua, yang menempatkan elemen terbesar kedua pada posisi terakhir berikutnya, dan seterusnya. Maka:

$$f(n) = (n-1) + (n-2) + \ldots + 2 + 1 = \frac{n(n-1)}{2} = \frac{n^2}{2} + O(n) = O(n^{2)}$$

Dengan kata lain, waktu untuk menjalankan algoritma bubble sort proporsional terhadap n^2, dimana n adalah jumlah data.

Pencarian, Linear Search

Misalkan DATA adalah kumpulan elemen data dalam memori, dan misalkan informasi ITEM diberikan. Pencarian mengacu pada operasi untuk mencari lokasi LOC dari ITEM dalam DATA atau mencetak pesan bahwa ITEM tidak terdapat dalam DATA. Pencarian dikatakan sukses apabila ITEM ada dalam DATA, dan gagal apabila sebaliknya.

Biasanya, penambahan suatu ITEM ke dalam DATA dilakukan setelah pencarian yang gagal untuk ITEM dalam DATA. Algoritma search and insertion lebih sering digunakan daripada hanya algoritma pencarian.

Ada beberapa algoritma pencarian. Pemilihan algoritma biasanya tergantung bagaimana cara informasi dalam DATA diorganisasikan. Berikut akan dibahas dua algoritma pencarian, pertama algoritma linear search dan berikutnya algoritma binary search.

Kompleksitas algoritma pencarian diukur berdasarkan jumlah f(n) perbandingan yang dibutuhkan untuk mencari ITEM dalam DATA dengan n elemen.

Linear Search

Misalkan DATA adalah array linear dengan n elemen. Tidak ada informasi lain yang berkaitan dengan DATA. Cara yang paling intuitif adalah membandingkan ITEM yang dicari dengan tiap ITEM dalam DATA satu per satu. Pertama kita uji apakah DATA[1] = ITEM, kemudian kita uji apakah DATA[2] = ITEM dan seterusnya. Metode ini akan menjelajahi DATA secara berurutan untuk mencari lokasi ITEM, ini yang disebut dengan Linear Search atau Sequential Search.

Untuk menyederhanakan masalah, pertama kita isi ITEM dengan DATA[N+1], posisi berikutnya dari elemen terakhir DATA. Maka akan didapat LOC = N + 1 dimana LOC dinotasikan sebagai lokasi dimana ITEM pertama kali ditemukan dalam DATA, menandakan bahwa pencarian itu gagal. Kegunaan dari penugasan awal ini adaalh untuk menghindari pengulangan pengujian apakah kita sudah atau belum mencapai akhir dari array DATA.

```
Algoritma (Linear Search) LINEAR(DATA, N, ITEM,
          LOC)
          Data adalah array linear dengan N
          elemen, dan ITEM adalah informasi yang
          diberikan. Algoritma ini mencari
          lokasi LOC ITEM dalam DATA, atau
          menentukan LOC = 0 jika pencarian
          gagal.

          1. [Insert ITEM pada bagian akhir DATA]
             Set DATA[N+1]:= ITEM.
          2. [Inisialisasi Counter] Set LOC:= 1.
          3. [Pencarian ITEM]
```

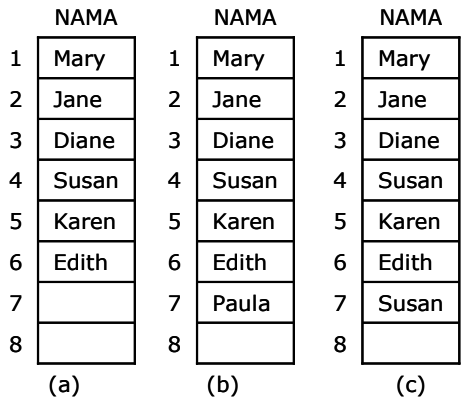
```
    Repeat While DATA[LOC] ≠ ITEM:
        Set LOC:= LOC + 1.
    [End of loop]
4. [Berhasil?] if LOC = N + 1, then Set
    LOC:= 0.
5. Exit.
```

Perhatikan, step 1 menjamin perulangan pada step 3 pasti berakhir. Tanpa step 1, maka pernyataan Repeat pada step 3 harus diubah menjadi:

```
Repeat While LOC ≤ N and DATA[LOC] ≠ ITEM:
```

Dilain pihak, pada penggunaan step 1, harus ada jaminan tersedianya lokasi memori yang belum terpakai pada bagian akhir DATA.

Contoh:
Perhatikan array NAMA pada gambar di bawah, dimana n = 6.



Gambar 6. 5. Array Nama

a. Misalkan kita ingin mengetahui apakag Paula ada dalam array, jika ada, dimana. Algoritma di atas secara temporer menempatkan Paula pada akhir array, seperti pada **Error!**

Reference source not found.b, dengan mengatur NAME[7] = Paula. Kemudian algoritma mencari dalam array dari atas ke bawah. Karena Paula ditemukan dalam NAMA[N+1], berarti Paula tidak terdapat dalam array.

b. Misalkan kita ingin mengetahui apakah Susan ada dalam array, jika ada, tentukan letaknya. Algoritma secara temporer menempatkan Susan pada bagian akhir array, lihat **Error! Reference source not found.**, dengan menempatkan NAMA[7] = Susan. Kemudian algoritma akan mencari dalam array mulai atas sampai bawah. Karena Susan ditemukan dalam NAMA[4] (dimana $4 \leq n$), berarti Susan ada dalam array.

Kompleksitas Algoritma Linear Search

Seperti dijelaskan di atas, kompleksitas algoritma pencarian diukur berdasarkan jumlah f(n) perbandingan untuk mencari ITEM dalam DATA yang mengandung n elemen. Dua hal yang terpenting untuk dipertimbangkan adalah average case dan *worst case*.

Jelas bahwa *worst case* ada jika pencarian dilakukan melalui seluruh DATA, misalkan ITEM tidak terdapat dalam DATA. Dalam hal ini, algoritma memerlukan $f(n) = n + 1$ perbandingan. Maka, dalam *worst case*, waktu yang diperlukan proporsional terhadap n.

Binary Search

Misalkan DATA adalah array yang diurutkan dari kecil ke besar berdasarkan angka atau alfabet. Ada algoritma pencarian yang sangat efisien, yang disebut binary search, yang dapat digunakan untuk mencari lokasi LOC dari ITEM informasi yang diberikan dalam DATA.

Misalkan kita ingin mencari lokasi beberapa nama dalam buku telepon. Jelas, kita tidak akan menggunakan pencarian linear, tapi kita akan membuka bagian tengah buku untuk melihat di bagian mana nama akan di cari. Kemudian kita membuka bagian tengah dari bagian yang kita cari atau seperempat bagian buku, dan seterusnya.

Algoritma binary search yang diterapkan pada array DATA akan bekerja sebagai berikut. Pada tiap tahap dalam algoritma, pencarian ITEM akan dikurangi menjadi satu segmen DATA:

```
DATA[BEG], DATA[BEG + 1], DATA[BEG + 2], …,
DATA[END]
```

Perhatikan bahwa variable BEG dan END dinotasikan sebagai lokasi awal dan lokasi akhir di bagian segmen yang terpilih. Algoritma membandingkan ITEM dengan elemen tengah DATA[MID] dari suatu segmen, dimana MID diperoleh dengan MID = INT((BEG + END)/2). Jika DATA[MID] = ITEM, maka pencarian berhasil dan kita nyatakan LOC:= MID, selain itu segmen baru dari DATA diperoleh dengan:

a. Jika ITEM < DATA[MID], maka ITEM ada pada bagian kiri segmen:
   DATA[BEG], DATA[BEG+1],..., DATA[MID-1].
   Jadi kita reset nilai END:= MID – 1 dan pencarian dimulai kembali.
b. Jika ITEM > DATA[MID], maka ITEM ada pada bagian kanan segmen:
   DATA[MID] + 1, DATA[MID + 1],..., DATA[END].
   Jadi kita reset nilai BEG:= MID + 1 dan pencarian dimulai kembali.

Awalnya, kita mulai dengan seluruh array DATA, misalnya kita mulai BEG = 1 dan END = n atau lebih umumnya, dengan BEG = LB dan END = UB.

Jika ITEM tidak ada dalam DATA, maka biasanya kita akan memperoleh END < BEG. Kondisi ini menandakan bahwa pencarian gagal, dan dalam hal ini kita nyatakan LOC = NULL.

Berikut adalah algoritma binary search:

```
Algoritma (Binary Search) BINARY(DATA, LB, UB,
          ITEM, LOC)
```

```
DATA adalah array terurut dengan batas
bawah LB dan batas atas UB dan
informasi ITEM yang diberikan.
Variable BEG, END, dan MID dinotasikan
sebagai lokasi awal, akhir dan tengah
dari segmen dalam elemen DATA.
Algoritma ini mencari lokasi LOC dari
ITEM dalam DATA atau menyatakan LOC =
NULL.

1. [Inisialisasi] Set BEG:= LB, END:=
   UB dan MID = INT((BEG+END)/2).
2. Repeat Step 3 and 4 whilw BEG ≤ END
   dan DATA[MID] ≠ ITEM:
3.      If ITEM < DATA[MID] Then:
              Set END:= MID – 1.
        Else
              Set BEG:= MID + 1.
        [End of if structure]
4.      Set MID:= INT((BEG + END)/2)
   [End of Step 2 loop]
5. If DATA[MID] = ITEM, then:
        Set LOC:= MID.
   Else:
        Set LOC:= NULL.
   [End of if structure]
6. Exit.
```

Contoh:

Misalkan DATA adalah array terurut 13 elemen dengan data sebagai berikut:

DATA: 11, 22, 30, 33, 40, 44, 55, 60, 66, 77, 80, 88, 99

Kita akan menerapkan binary search pada array DATA untuk nilai ITEM yang berbeda.

a. Misalkan ITEM = 40. Pencarian ITEM dalam array DATA dijelaskan pada **Error! Reference source not found.**, dimana nilai DATA[BEG] dan DATA[END] pada tiap tahap ditandai dengan background biru dan nilai DATA[MID] ditandai dengan background abu-abu. Secara khusus, BEG, END dan MID akan mempunyai nilai sebagai berikut:
    1. Awaknya, BEG = 1 dan END = 13. Maka MID = INT((1 + 13)/2) = 7, jadi DATA[MID] = 55.
    2. Karena 40 < 55, nilai END akan diubah menjadi END = MID – 1 = 6. Maka MID = INT((1 + 6)/2) = 3, jadi DATA[MID] = 30.
    3. Karena 40 > 30, nilai BEG akan diubah menjadi BEG = MID + 1 = 4. Maka MID = INT((4 + 6)/2) = 5, jadi DATA[MID] = 40.

    Kita menemukan lokasi ITEM yaitu LOC = MID = 5.

(1)	11	22	30	33	40	44	55	60	66	77	80	88	99
(2)	11	22	30	33	40	44	55	60	66	77	80	88	99
(3)	11	22	30	33	40	44	55	60	66	77	80	88	99

Gambar 6. 6. LOC Item

b. Misalkan ITEM = 85. Pencarian menggunakan binary search digambarkan pada **Error! Reference source not found.**. Variabel BEG, END dan MID akan memiliki nilai sebagai berikut:
    1. Awalnya, BEG = 1, END = 13, MID = 7 dan DATA[MID] = 55.
    2. Karena 85 > 55, maka nilai BEG akan diubah menjadi BEG:= MID + 1 = 8. MID:= INT((8 + 13)/2) = 10, dan DATA[MID] = 77.

3. Karena 85 > 77, maka nilai BEG akan diubah menjadi BEG:= MID + 1 = 11. MID:= INT((11+13)/2) = 12, dan DATA[MID] = 88.
4. Karena 85 < 88, maka nilai END akan diubah menjadi END:= MID – 1 = 11. MID:= INT((11+11)/2) = 11, dan DATA[MID] = 80. Perhatikan bahwa nilai BEG = END = MID = 11.

Karena 85 > 80, nilai BEG akan diubah menjadi BEG = MID + 1 = 12. Tapi sekarang BEG > END. Karena itu ITEM berarti tidak ada dalam DATA. Berarti pencarian gagal.

(1)	11	22	30	33	40	44	55	60	66	77	80	88	99
(2)	11	22	30	33	40	44	55	60	66	77	80	88	99
(3)	11	22	30	33	40	44	55	60	66	77	80	88	99
(4)	11	22	30	33	40	44	55	60	66	77	80	88	99

Gambar 6. 7. Data

Kompleksitas Algoritma Binary Search

Kompleksitas diukur berdasarkan banyaknya f(n) perbandingan untuk mencari ITEM dalam DATA dimana DATA mengandung n elemen. Perhatikan, pada tiap perbandingan ukuran sampel dikurangi setengahnya. Karena itu kita akan memerlukan paling banyak f(n) perbandingan untuk mencari ITEM dimana 2f(n) > n atau secara eqivalen f(n) = $\lfloor$Log2 n$\rfloor$ + 1.

Waktu untuk *worst case* kira-kia sama dengan log2 n. Sedangkan untuk average case kira-kira sama dengan waktu untuk menjalankan *worst case*.

Contoh:

Misalkan DATA mengandung 1.000.000 elemen. Perhatikan bahwa 210 = 1024 > 1000 dan 220 > 10002 = 1.000.000.

Maka, dengan menggunakan algoritma binary search, kita hanya memerlukan 20 perbandingan untuk mencari lokasi suatu item dalam array dengan 1.000.000 elemen.

Keterbatasan Algoritma Binary Search

Algoritma binary search adalah algoritma yang sangat efisien (karena hanya memerlukan 20 perbandingan untuk mencari data yang mempunyai 1 juta elemen). Kenapa kita masih memerlukan algoritma pencarian lain? Perhatikan bahwa algoritma binary search memerlukan dua kondisi yaitu:

1. Daftar harus terurut.
2. Kita harus mempunyai akses langsung ke bagian tengah data.

Ini berarti kita harus mempunyai array yang terurut untuk menyimpan data. Akan tetapi menjaga data dalam keadaan terurut sangatlah sulit terutama jika terjadi banyak penghapusan dan penyisipan. Karena itu untuk beberapa situasi, kita harus menggunakan struktur data yang lain, seperi linked list atau binary search tree untuk menyimpan data.

Array Multidimensi

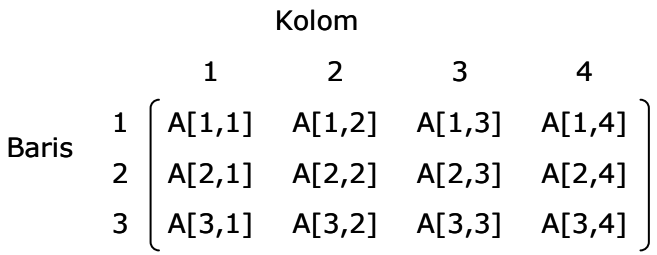
Array linear yang kita bahas selama ini dikenal dengan array satu dimensi, karena tiap elemen dalam array direferensi menggunakan satu subscript. Beberapa bahasa pemrograman membolehkan penggunaan array dua dimensi dan tiga dimensi, dimana elemen-elemennya direferensikan menggunakan dua dan tiga subscript.

Array Dua Dimensi

Array dua dimensi m x n adalah kumpulan m x n elemen data dimana tiap elemen dibedakan menggunakan sepasang integer (seperti j, k) yang disebut subscript. Elemen A dengan subscript pertama j dan subscript kedua k akan dinotasikan menggunakan AJ,K atau A[J, K]. Array dua dimensi disebut matriks dalam matematika dan table dalam

aplikasi bisnis, karena itu array dua dimensi kadang-kadang disebut array matriks.

Array dua dimensi m x n array A digambarkan dengan m baris dan n kolom dan elemen A[J, K] ada pada perpotongan baris ke J dan kolom ke K. Gambar di bawah menunjukkan array dua dimensi A dengan 3 baris dan 4 kolom.



Gambar 6. 8. Array Dua Dimensi

Contoh:

Misalkan terdapat satu kelas dengan 25 mahasiswa menerima 4 test berbeda. Diasumsikan mahasiswa diberi nomor 1 sampai dengan 25. Nilai test dapat disimpan menggunakan 25 x 4 array matriks SCORE (lihat **Error! Reference source not found.**). SCORE[K, L] menunjukkan nilai test ke L dari mahasiswa ke K. Khususnya, pada baris kedua array SCORE[2,1], SCORE[2,2], SCORE[2,3], dan SCORE[2,4] menunjukkan empat nilai test mahasiswa kedua.

Mahasiswa	Test 1	Test 2	Test 3	Test 4
1	84	73	88	81
2	95	100	88	96
3	72	66	77	72
...	...	...	...	...
25	78	82	70	85

Gambar 6. 9. Contoh Kasus Array Dua Dimensi

Misalkan A adalah array dua dimensi dengan m x n array. Dimensi pertama dari A memiliki indeks 1, 2,..., m dengan batas bawah 1 dan

batas atas m, dan dimensi kedua memiliki indeks 1, 2,..., n dengan batas bawah 1 dan batas atas n. Panjang dimensi array A adalah perkalian m x n atau disebut juga dengan ukuran array A.

Beberapa bahasa pemrograman membolehkan deklarasi variabel array dua dimensi dengan menggunakan batas bawah bukan 1. Bagaimanapun, kumpulan indeks untuk tiap dimensi merupakan angka yang berurutan dari batas bawah sampai batas atas. Panjang masing-masing dimensi dapat dihitung dengan rumus Lenth = batas atas – batas bawah + 1. Jika tidak dikatakan lain kita akan menggunakan angka 1 sebagai indeks batas bawah array.

Tiap bahasa pemrograman mempunyai aturan tersendiri untuk mendeklarasikan array multidimensi. Seperti array linear, semua elemen dalam array harus menggunakan tipe data yang sama. Misalkan kita memiliki array dua dimensi DATA dengan 4 x 8 elemen dengan tipe data real. Kita dapat mendeklarasikannya dengan cara:

```
FORTRAN:    REAL DATA(4, 8)
PL/1:       DECLARE DATA(4, 8) FLOAT;
PASCAL:     VAR DATA: ARRAY[1..4, 1..8] OF REAL;
```

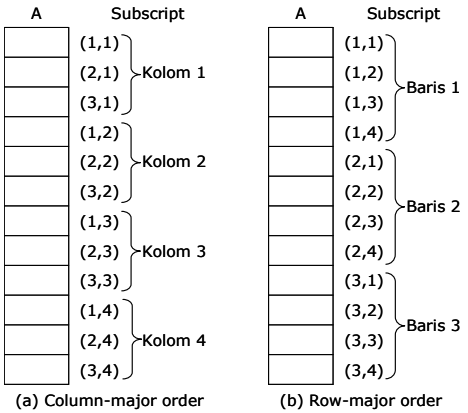
Perhatikan, pada PASCAL angka batas bawah tetap dicantumkan walaupun indeks batas bawahnya 1.

Representasi Array Dua Dimensi dalam Memory

Misalkan A adalah array dua dimensi dengan ukuran m x n. Walaupun A digambarkan sebagai array berbentuk segiempat dengan m baris dan n kolom, array akan digambarkan di memory menggunakan m x n blok lokasi memori yang berurutan. Secara khusus, bahasa pemrograman akan menyimpan array A dengan cara:

1. Kolom per kolom, sehingga ini disebut dengan column-major order, atau
2. bari per baris, atau disebut row-major order.

Gambar di bawah menunjukkan dua cara tersebut. Perlu ditekankan bahwa representasi tertentu yang digunakan terganting dengan bahasa pemrograman bukan pengguna.



Gambar 6. 10. Representasi Array Dua Dimensi

Ingat, untuk array linear LA, komputer tidak perlu menyimpan alamat LOC(LA[K]) untuk tiap elemen LA[K] dari LA, tetapi cukup menyimpan Base(LA) yaitu alamat elemen pertama dari LA. Komputer menggunakan rumus LOC(LA[K]) = Base(LA) + w(K – 1) untuk mencari alamat dari LA[K]. w adalah jumlah word per sel memori untuk array LA, dan 1 adalah batas bawah dari indeks LA.

Hal yang sama juga berlaku untuk sembarang array dua dimensi dengan m x n elemen. Komputer menyimpan jejak Base(A), alamat dari elemen pertama A[1,1] dari A dan menghitung alamat LOC(A[J,K]) menggunakan rumus:

a. Colum-major Order
   LOC(A[J,K]) = Base(A) + w[M(K-1) + (J-1)]
b. Row-major order
   LOC(A[J,K]) = Base(A) + w[N(J-1) + (K-1)]

w menunjukkan jumlah word per lokasi memori untuk array A.

Contoh:

Perhatikan SCORE, array matriks 25 x 4 pada Contoh. Misalkan Base(SCORE) = 200 dan w = 4 word per sel memory. Lebih jauh, misalkan bahasa pemrograman menyimpan array dua dimensi menggunakan row-major order. Maka alamat SCORE[12,3], nilai test ketiga dari mahasiswa ke 12 adalah:

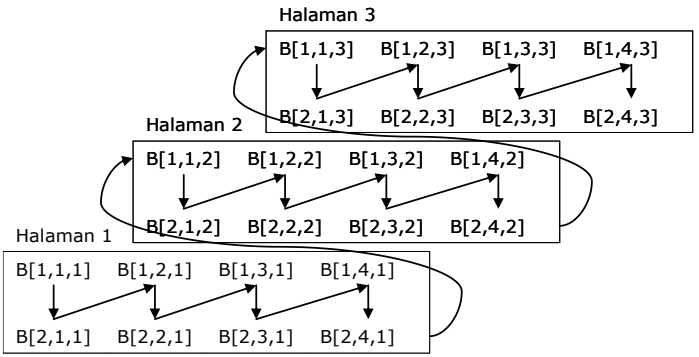
LOC(SCORE[12,3]) = 200 + 4(4(12-1)+(3-1)) = 200 + 4(46) = 384

Array Multidimensi Umum

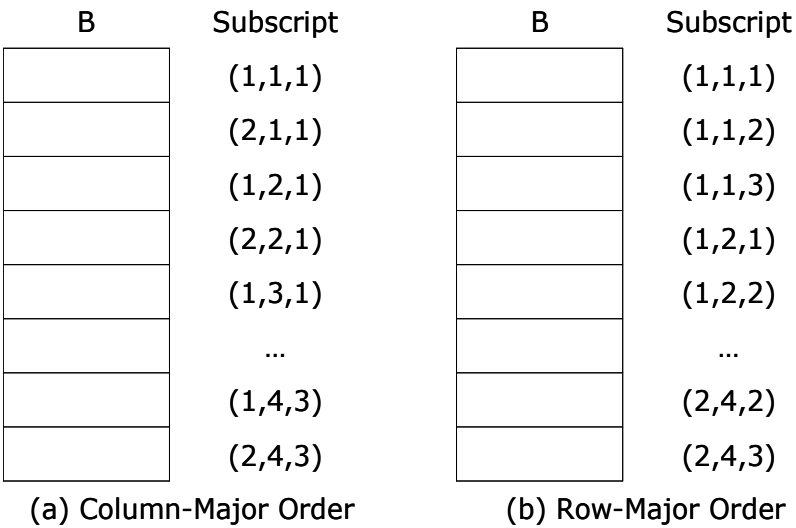
Array multidimensi umum didefinisikan sama. Secara khusus, array B suatu array n-dimensi m1 x m2 x... x mn adalah koleksi dari m1. m2... mn elemen data dimana tiap elemen ditunjukkan oleh suatu daftar n integer seperti K1, K2,.., Kn yang disebut subscript. Elemen B dengan subscript K1, K2,..., Kn akan dinotasikan sebagai BK1,K2,...,Kn atau B[K1,K2,...,KN]. Array akan disimpan di memory dalam lokasi yang berurutan. Secara khusus, bahasa pemrograman akan menyimpan array B menggunakan row-major order dan column-major order.

Contoh:

Misalkan B adalah array tiga dimensi dengan ukuran 2 x 4 x 3. Maka B mengandung 2 x 4 x 3 = 24 elemen. Elemen-elemen array B biasanya digambarkan seperti gambar di bawah. Terbentuk dari tiga lapisan, yang disebut halaman (pages), dimana tiap halaman terdiri dari 2 x 4 matriks array. Ketiga subscript dalam array disebut baris, kolom dan halaman. B disimpan di memory dengan dua cara.



Gambar 6. 11. Array Multidimensi



Gambar 6. 12. Array Multidimensi

Maka alamat LOC(C[K1, K2,..., KN]) dari sembarang elemen C dapat diperoleh dari rumus:

Base(C) + w[((...(ENLN-1+EN-1)LN-2)+...+E3)L2+E2)L1+E1]

Atau

Base(C) + w[(...((E1L2+EN-1)L3+E3)L4+...+EN-1)LN+EN]

Tergantung apakah C disimpan dalam column-major order atau row-major order, Base(C) menunjukkan alamat elemen pertama C dan w adalah jumlah words per lokasi memory.

Contoh:

Misalkan MAZE adalah array tiga dimensi dan dideklarasikan menggunakan MAZE(2:8,-4:1,6:10). Maka panjang ketiga dimensinya masing-masing:

L1 = 8 – 2 + 1 = 7, L2 = 1 – (-4) + 1 = 6, L3 = 10 – 6 + 1 = 5

Berarti, MAZE mengandung L1 * L2 * L3 = 7 * 6 * 5 = 210 elemen.

Misalkan suatu bahasa pemrograman menyimpan MAZE dalam memorynya menggunakan row-major order, Base(MAZE) = 200 dan w = 4 words per sel memori. Maka, alamat untuk MAZE[5, -1, 8] diperoleh dengan cara sebagai berikut:

Indeks effective dari subscript adalah:
E1 = 5 – 2 = 3, E2 = -1 – (-4) = 3, E3 = 8 – 6 = 2

Menggunakan persamaan untuk row-major order, kita menemukan:

E1L2 = 3 * 6 = 18
E1L2 + E2 = 18 + 3 = 21
(E1L2 + E2)L3 = 21 * 5 = 105
(E1L2 + E2)L3 + E3 = 105 + 2 = 107

Maka,

LOC(MAZE[5,-1,8]) = 200 + 4(107) = 200 + 428 = 628.

6.2. Pointer

Misalkan DATA adalah sembarang array. Suatu variable P disebut suatu pointer jika P menunjuk ke salah satu elemen dalam DATA, sebagai contoh, jika P mengandung alamat dari satu elemen dalam DATA. Analoginya, suatu array PTR disebut array pointer jika tiap elemen dari PTR adalah pointer. Pointer dan array pointer digunakan untuk menjembatani pengolahan informasi dalam DATA. Bagian berikut ini membahas penggunaan pointer dalam beberapa contoh khusus.

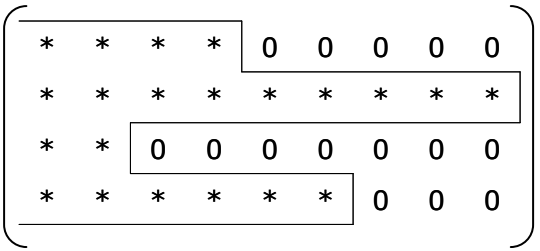
Misalkan, suatu organisasi membagi anggotanya menjadi empat group, dimana setiap group terdiri dari daftar terurut secara alphabet untuk anggota yang tinggal di area tertentu. Gambar di bawah menunjukkan daftar tersebut. Perhatikan disana ada 21 orang dan group masing-masing berisi 4, 9, 2, dan 6 orang.

Group 1	Group 2	Group 3	Group 4
Evans	Conrad	Davis	Baker
Harris	Felt	Segal	Cooper
Lewis	Glass		Ford
Shaw	Hill		Gray
	King		Jones
	Penn		Reed
	Silver		
	Troy		
	Wagner		

Gambar 6. 13. Daftar Pointer

Misalkan daftar tersebut disimpan dalam memory dengan menjaga urutan pada masing-masing group. Kita bisa menggunakan array dua dimensi dengan 4 x n elemen dimana tiap baris berisi suatu group atau array dua dimensi dengan n x 4 elemen dimana tiap kolom berisi suatu

group. Walaupun struktur data ini membolehkan kita mengakses tiap individual group, akan tetapi akan membuang banyak ruang apabila tiap group mempunyai jumlah anggota yang sangat bervariasi. Secara khusus, **Error! Reference source not found.** akan memerlukan paling sedikit 36 elemen dari array 4 x 9 atau 9 x 4 untuk menyimpan 21 nama, yang berarti hampir dua kali dari yang diperlukan. Gambar di bawah menunjukan representasi array 4 x 9, dimana * menunjukkan data dan 0 menunjukkan lokasi yang tidak terpakai.

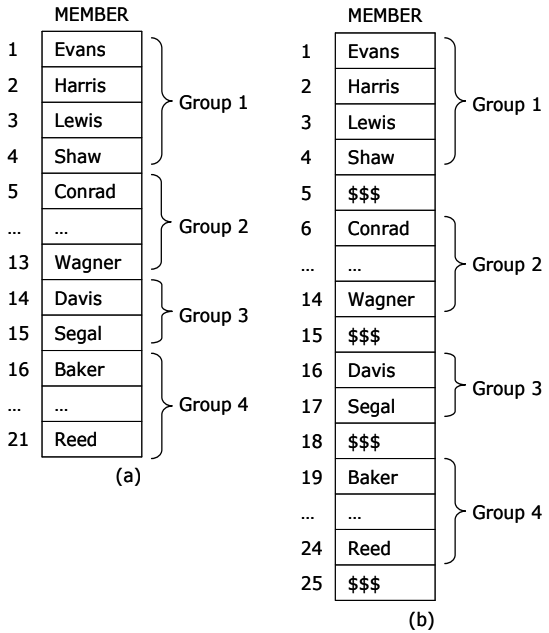


Gambar 6. 14. Matriks Pointer

Cara lain untuk menyimpan daftar anggota dapat dilihat pada gambar a di bawah. Daftar anggota diletakkan dalam array linear, satu group disusun sesudah group lainnya. Jelas cara ini sangat efisien. Juga, seluruh daftar dapat diproses dengan mudah, kita bisa mencetak seluruh nama dalam daftar. Di lain sisi, tidak ada cara untuk mengakses group tertentu, sebagai contoh, tidak ada cara untuk mencari dan mencetak hanya nama-nama yang ada dalam group 3.

Versi modifikasi metode di atas dapat dilihat pada gambar b di bawah. Nama di masukkan ke dalam array linear, group per group, ditambah dengan beberapa sentinel atau penanda, dalam hal ini $$$ digunakan untuk menandai akhir dari tiap group. Metode ini mengggunakan beberapa sel memori tambahan, satu untuk tiap group, tapi kita sekarang dapat dengan mudah mengakses group tertentu. Sebagai contoh, seorang programmer dapat mencari dan mencetak seluruh nama pada group 3 dengan cara mencari nama-nama yang ada setelah sentinel kedua dan sebelum sentinel ketiga. Kelemahan utama

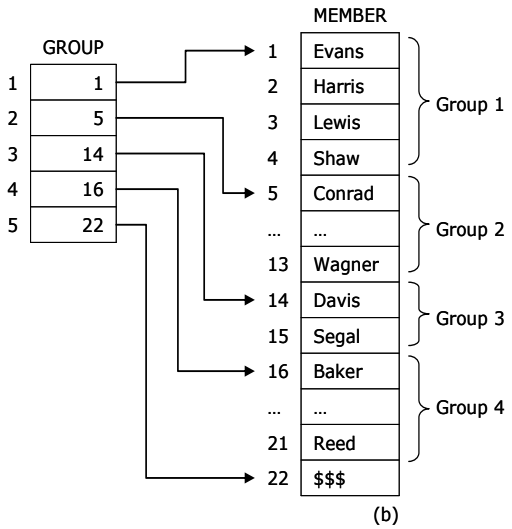
dari representasi ini adalah daftar masih harus dijelajahi dari awal untuk mengenali group 3.



Gambar 6. 15. Daftar Pointer

Array Pointer

Struktur data pada gambar a dan b di atas apat dimodifikasi sehingga setiap group dapat diindeks. Hal ini bisa dilaksanakan menggunakan array pointer yang mengandung lokasi dari group yang berbeda atau secara khusus, lokasi pertama dari elemen pada masing-masing group. Gambar di bawah menunjukkan gambar a dan b di atas yang telah dimodifikasi. Perhatikan GROUP[L] dan GROUP[L+1]-1 berisi masing-masing elemen awal dan akhir dari group L. (Perhatikan GROUP[5] menunjukkan sentinel dari daftar dan GROUP[5]-1 memberikan kita lokasi dari elemen terakhir group 4.)



Gambar 6. 16. Array Pointer

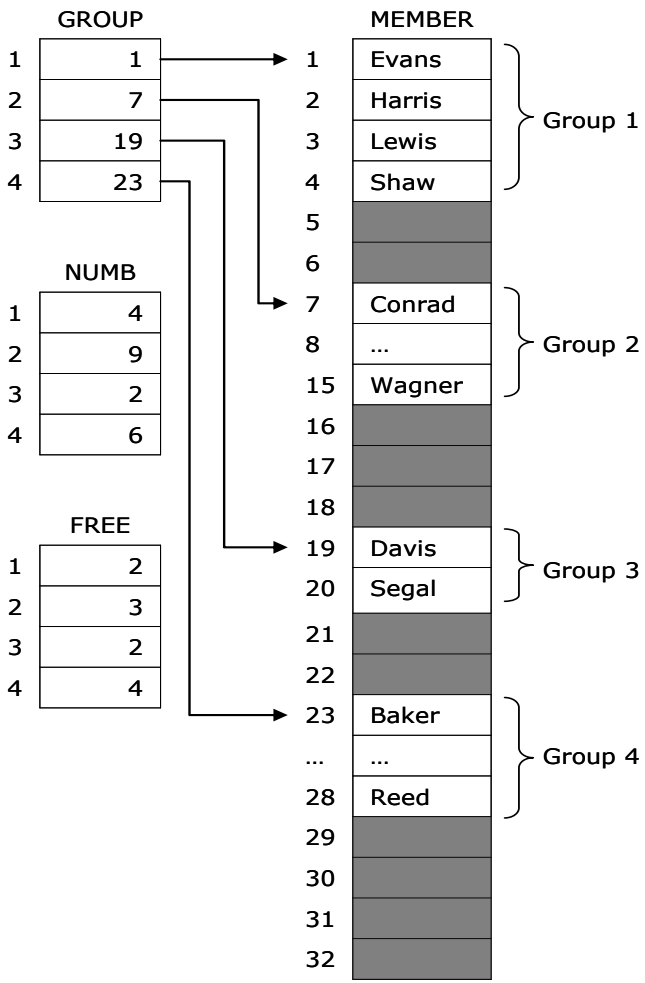
Contoh:

Misalkan kita hanya ingin mencetak nama-nama yang ada di group ke-L pada gambar di bawah, dimana nilai L adalah bagian dari input. Karena GROUP[L] dan GROUP[L+1]-1 mengandung masing-masing lokasi awal dan akhir nama dalam group L, modul berikut ini dapat digunakan untuk menyelesaikan masalah di atas:

```
1. Set FIRST:= GROUP[L] and LAST:= GROUP[L+1]-
   1
2. Repeat For K = FIRST to LAST
         Write: MEMBER[K]
   [End of loop]
3. Return.
```

Variasi lain dari struktur data pada gambar di atas ditunjukkan oleh gamabar di bawah, dimana memori yang tidak terpakai ditandai oleh warna abu-abu. Perhatikan, sekarang terdapat beberapa ruang kosong antar group. Karena itu, elemen baru dapat dengan mudah disisipkan

tanpa harus memindah elemen-elemen pada group lain. Menggunakan struktur data ini, kita memerlukan array NUM yang akan memberikan jumlah elemen tiap group. Perhatikan, GROUP[K+1] – GROUP[K] adlaah jumlah total ruang yang tersedia untuk Group K, karenanya FREE[K] = GROUP[K+1] – GROUP[K] – NUMB[K] adalah jumlah sel yang kosong dari group K.



Gambar 6. 17. Array Pointer

Contoh:

Misalkan, kita ingin hanya mencetak seluruh nama yang ada pada group L, dimana L adalah bagian dari input, akan tetapi sekarang group disimpan seperti pada. Perhatikan bahwa GROUP[L] dan GROUP[L] + NUMB[L] – 1 berisi masing-masing alamat pertama dan alamat terakhir dari group L. Berikut ini adalah modul yang dapat digunakan untuk memecahkan masalah di atas.

```
1. Set FIRST:= GROUP[L] and LAST:= GROUP[L] +
   NUMB[L] – 1
2. Repeat For K = FIRST to LAST
        Write: MEMBER[K]
   [End of loop]
   Return.
```

6.3. Record

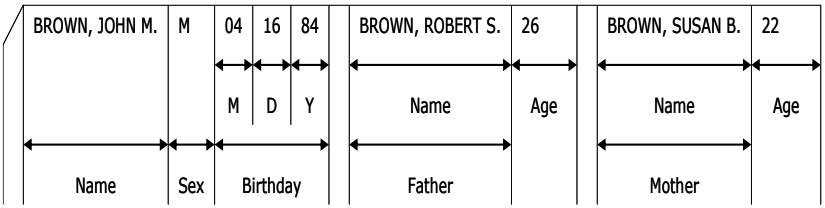
Kumpulan data biasanya diorganisasikan ke dalam bentuk hirarki fields, record dan file. Secara khusus, record adalah kumpulan item data yang berkaitan yang disebut field atau atribut dan file adalah kumpulan record yang sejenis. Tiap item data dapat berupa group item yang terdiri dari beberapa subitem, yang apabila subitem itu diuraikan akan disebut item dasar atau atom atau scalar. Nama yang diberikan untuk beberapa item data disebut identifier.

Walaupun record adalah kumpulan item data, record berbeda dengan array linear pada beberapa hal sebagai berikut:

a. Record dapat berupa kumpulan data yang non homogen, sebagai contoh, record dapat mempunyai tipe data yang berbeda.
b. Item data dalam record diindeks menggunakan nama atributnya.

Berdasarkan hubungan antara item group dengan subitem, item data dalam record membentuk struktur hirarki yang dapat dijelaskan

menggunakan tingkatan angka, seperti yang digambarkan pada contoh di bawah



Gambar 6. 18. Contoh Record

Contoh:

Misalkan suatu rumah sakit menyimpan record setiap kelahiran bayi dimana tiap record mengandung item data sebagai berikut: Nama, Jenis Kelamin, Tanggal Lahir, Ayah dan Ibu. Lebih jauh, Tanggal Lahir adalah group item dengan subitem Tanggal, Bulan dan Tahun, demikian juga dengan Ayah dan Ibu merupakan group item dengan subitem Nama dan Umur. **Error! Reference source not found.** menunjukkan bagaimana suatu record ditampilkan.

Struktur record di atas biasanya dijelaskan sebagai beikut:

1 Newborn
- 2 Name
- 2 Sex
- 2 Birthday
  - 3 Month
  - 3 Day
  - 3 Year
- 2 Father
  - 3 Name
  - 3 Age
- 2 Mother
  - 3 Name
  - 3 Age

Angka yang ada pada bagian kiri menunjukkan nomor level. Perhatikan tiap item group diikuti oleh subitemnya dan level subitem bernomor lebih tinggi 1 dari level item group.

Beberapa identifier dalam struktur record dapat juga mengacu pada elemen array. Misalkan baris pertama pada struktur di atas diganti dengan

1 Newborn(20)

Ini berarti file terdiri dari 20 record, dan notasi subscript biasa digunakan untuk membedakan antar record dalam file. Maka, kita dapat menulis Newborn1, Newborn2,...., NewbornN atau Newborn[1], Newborn[2], Newborn[3],..., Newborn[N].

Contoh:

Record mahasiswa dari suatu kelas disusun sebagai berikut:

```
1 Student(20)
        2 Name
                3 Last
                3 Last
                3 MI (Middle Initial)
        2 Test(3)
        2 Final
        2 Grade
```

Identifier Student(20) menandakan terdapat 20 mahasiswa. Identifier Test(3) menandakan terdapat 3 test tiap mahasiswa. Perhatikan terdapat 8 item dasar per mahasiswa, karena identifier Test dihitung tiga kali. Berarti, terdapat 160 item dasar dalam seluruh struktur Student.

Mengindeks Item dalam Record

Misalkan kita ingin mengakses beberapa item data dalam record. Dalam beberapa kasus, kita tidak depat dengan mudah menuliskan nama data item karena nama yang sama ada pada tempat lain dalam record. Sebagai contoh, Age muncul dua kali dalam record di Contoh. Maka, untuk membedakan antar item tertentu, kita harus mengkualifikasi nama menggunakan nama item group tertentu. Kualifikasi ini ditandai menggunakan titik untuk memisahkan item group dengan subitemnya.

Contoh:

a. Perhatikan struktur record Newborn pada Contoh. Sex dan Year tidak perlu kualifikasi, karena keduanya mengacu pada item yang unik dalam struktur. Di lain sisi, misalkan kita ingin mengacu pada umur ayah, maka kita dapat melakukannya dengan:

   Newborn.Father.Age atau Father.Age.

b. Misalkan baris pertama dalam record pada Contoh diganti dengan

   1 Newborn(20).

   Newborn didefinisikan menjadi file dengan 20 record. Maka tiap item secara otomatis menjadi array 20 elemen. Beberapa bahasa pemrograman membolehkan item Sex untuk Newborn yang ke enam diakses dengan menuliskan Newborn.Sex[6] atau Sex[6]. Sejalan dengan itu, umur ayah dapat diakses dengan menuliskan Newborn.Father.Age[6] atau Father.Age[6]

c. Perhatikan struktur record Student pada Contoh. Karena Student dideklarasikan menjadi file dengan 20 mahasiswa, seluruh item secara otomatis menjadi array 20 elemen. Lebih jauh, Test menjadi array dua dimensi. Dalam hal ini, Test kedua untuk mahasiswa ke enam dapat direferensi menggunakan

   Student.Test[6, 2] atau Test[6, 2].

   Urutan subscript berkaitan dengan urutan kualifikasi identifier. Contohnya, Test[3,1] bukan mengacu pada test ketiga untuk mahasiswa pertama, tapi test pertama untuk mahasiswa ketiga.

Representasi Record di Memori; Array Paralel

Karena record mengandung data yang non-homogen, elemen dari record tidak dapat disimpan dalam suatu array. Beberapa bahasa pemrograman seperti PL/1, Pascal dan COBOL mempunyai struktur record yang menyatu dengan bahasanya.

Contoh:

Perhatikan struktur record Newborn pada contoh di atas. Kita dapat menyimpan record dalam PL/1 menggunakan deklarasi sebagai berikut, dimana didefinisikan suatu agregat data yang disebut structure:

```
DECLARE 1 NEWBORN,
            2 NAME CHAR(20),
            2 SEX CHAR(1),
            2 BIRTHDAY,
                  3 MONTH FIXED,
                  3 DAY FIXED,
                  3 YEAR FIXED,
            2 FATHER,
                  3 NAME CHAR(20),
```

```
        3 AGE FIXED,
    2 MOTHER,
        3 NAME CHAR(20),
        3 AGE FIXED;
```

Perhatikan variable SEX dan YEAR adalah variabel unik, karena itu untuk mereferensikannya tidak perlu dikualifikasikan. Di lain sisi, AGE tidak unik. Maka kita harus menggunakan FATHER.AGE atau MOTHER.AGE.

Misalkan suatu bahasa pemrograman tidak menyediakan struktur seperti pada PL/1, PASCAL dan COBOL. Diasumsikan record mengandung data yang non-homogen, record harus disimpan dalam variabel tersendiri, satu untuk tiap item data dasar. Di lain sisi, kita ingin menyimpan seluruh file, maka semua elemen data yang mempunyai identifier sama harus mempunyai tipe data yang sama. File harus disimpan di memori sebagai kumpulan array paralel, dimana elemen dalam array yang berbeda mempunyai subscript untuk record yang sama. Lihat contoh di bawah ini.

Contoh:

Misalkan suatu daftar anggota mengandung nama, umur, jenis kelamin dan telepon untuk masing-masing anggota. Kita dapat menyimpan file tersebut dalam empat array paralel NAME, AGE, SEX dan PHONE seperti pada gambar di bawah, untuk subscript K, elemen NAME[K], AGE[K], SEX[K] dan PHONE[K] milik record yang sama.

	NAME	AGE	SEX	PHONE
1	John Brown	28	Male	234-5186
2	Paul Cohen	33	Male	456-7272
3	Mary Davis	24	Female	777-1212
4	Linda Evans	27	Female	876-4478
5	Mark Green	31	Male	255-7654
...	...	...	...	...

Gambar 6. 19. Contoh Record

Contoh:

Perhatikan record NEWBORN pada Contoh. Kita dapat menyimpan file dalam 9 array linear yaitu NAME, SEX, MONTH, DAY, YEAR, FATHERNAME, FATHERAGE, MOTHERNAME dan MOTHERAGE, satu array untuk satu item data. Kita harus menggunakan nama variabel yang berbeda untuk nama dan umur masing-masing ayah dan ibu. Kita asumsikan bahwa array tersebut paralel, maka untuk subscript K, elemen NAME[K], SEX[K],..., MOTHERAGE[K] adalah milik record yang sama.

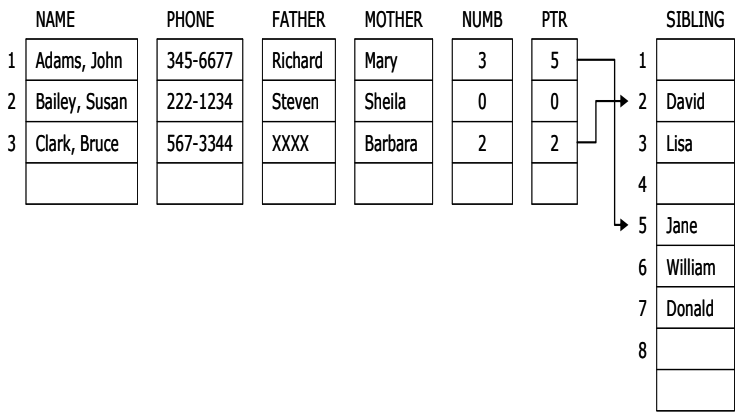
Record dengan Panjang Variabel Berbeda

Misalkan suatu sekolah dasar menyimpan record untuk tiap siswanya dengan item data: Nama, No Telepon, Ayah, Ibu, dan Saudara. Ayah, Ibu dan Saudara masing-masing mengandung nama ayah, nama ibu dan nama saudara siswa yang bersekolah ditempat yang sama. Berikut adalah contoh record:

```
Adams, John; 345-6677; Richard; Mary; Jane,
William, Donald
Bailey, Susan; 222-1234; Steven; Sheila; XXXX
Clark, Bruce; 567-3344; XXXX; Barbara; David,
Lisa
```

Disini XXXX berarti bahwa orang tuanya sudah meninggal atau tidak tinggal bersama siswa bersangkutan, atau siswa tidak mempunyai saudara yang sekolah di sekolah tersebut.

Contoh di atas menggunakan record dengan panjang variabel berbeda, karena elemen data saudara dapat berisi nol atau lebih nama. Salah satu cara untuk menyimpan file dalam array digambarkan pada gambar di bawah, dimana terdapat linear array NAME, PHONE, FATHER dan MOTHER untuk melayani empat item data pertama dalam record, dan array NUMB dan PTR masing-masing untuk nomor dan lokasi saudaranya dalam array SIBLING.



Gambar 6. 20. Contoh Record

6.4. Soal Latihan

1. Misalkan array linear AAA(5:50), BBB(-5:10) dan CCC(18).
   a. Tentukan jumlah elemen tiap array.
   b. Misalkan Base(AAA) = 300 dan w = 4 words per sel memori untuk AAA. Tentukan alamat AAA[15], AAA[35], AAA[55].
2. Perhatikan array linear NAMA pada **Error! Reference source not found.**
   a. Tentukan jumlah elemen yang harus dipindah jika Brown, Johnson dan Peters disisipkan pada NAMA pada tiga waktu yang berbeda.

b. Berapa banyak elemen yang dipindah jika tiga nama disisipkan pada saat yang bersamaan.
c. Bagaimana perusahaan telepon menangani penyisipan dalam direktori telepon.

	NAMA
1	Allen
2	Clark
3	Dickens
4	Edwards
5	Goodman
6	Hobbs
7	Irwin
8	Klein
9	Lewis
10	Morgan
11	Richards
12	Scott
13	Tucker
14	Walton

3. Misalkan array multidimensi A dan B dideklarasikan menggunakan A(-2:2, 2:22) dan B(1:8, -5:5, -10:5).
   a. Cari panjang tiap dimensi dan jumlah elemen dalam A dan B.
   b. Perhatikan elemen B[3, 3, 3] dalam B. Cari indeks efektif E1, E2, E3 dan alamat elemen, dengan asumsi Base(B) = 400 dan terdapat w = 4 word per sel memori.

BAB VII

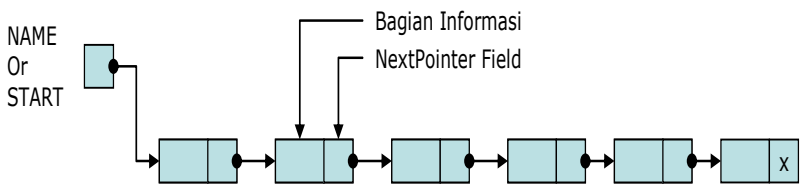
LINKED LIST

Capaian Pembelajaran:

1. Mampu menjelaskan penggunaan linked list
2. Mampu menjelaskan penggunaan representasi linked di memory
3. Mampu menjelajahi, mencari, menghapus dan menyisipkan elemen linked list.

Linked list atau *one-way list* adalah suatu kumpulan linear dari elemen data, yang disebut nodes, dimana urutan linearnya dibentuk menggunakan pointer. Tiap node dibagi menjadi dua bagian, bagian pertama berisi informasi mengenai elemen dan bagian kedua yang biasa disebut link field atau nextpointer field berisi alamat node berikutnya dalam list.

Gambar di bawah adalah diagram skematik suatu linked list dengan enam nodes. Tiap node digambarkan dengan dua bagian. Bagian kiri menggambarkan bagian informasi dari node, yang mungkin berisi seluruh record dari item data (misalkan NAMA, ALAMAT dan lain-lain). Bagian kanan mewakili nextpointer field dari node dan terdapat panah yang digambarkan menunjuk ke node berikutnya dari list. Pointer dari node terakhir mengandung nilai tertentu, yang disebut null pointer.

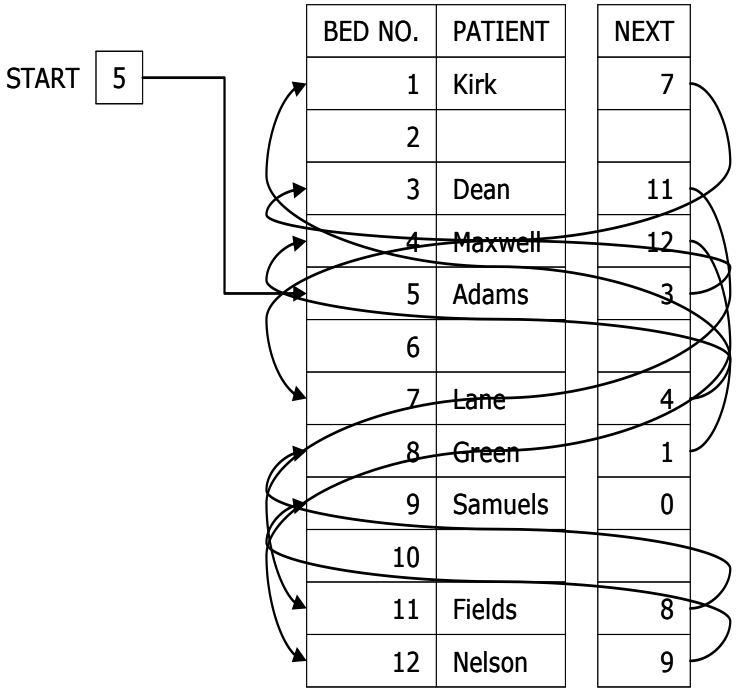


Gambar 7. 1. Linked List 6 Nodes

Dalam prakteknya, 0 atau angka negative digunakan sebagai null pointer. Null pointer dinotasikan dengan x dalam diagram, menandakan akhir dari list. Linked list juga mengandung list pointer variable, yang disebut START atau NAME, yang mengandung alamat node pertama dari list, karena itu ada panah yang digambar dari START ke node pertama. Jelasnya, kita hanya perlu alamat di START untuk melacak seluruh list. Pada situasi khusus, list bisa jadi tidak mempunyai node. List seperti ini disebut null list atau empty list dan dinotasikan menggunakan null pointer dalam variable START.

Contoh:

Sebuah rumah sakit memiliki 12 tempat tidur, 9 diantaranya telah ditempati pasien. Misalkan kita ingin membuat daftar pasien yang terurut secara alfabet. Daftar ini dilengkapi dengan pointer field, yang disebut NEXT. Kita menggunakan variabel START untuk menunjuk pasien pertama. Karena START berisi 5, maka pasien pertama, Adam menempati tempat tidur nomor 5. Kemudian Pointer Adam menunjuk ke 3, maka Dean, pasien berikutnya menempati ranjang nomor 3 dan seterusnya. Entry Pasien terakhir (Samuel) berisi null pointer yang ditandai dengan 0.

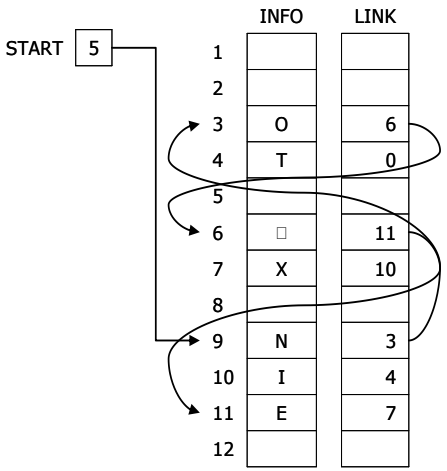


Gambar 7. 2. Contoh Linked List

7.1. Representasi Linked List di Memori

Misalkan B adalah suatu linked list. Pertama, kita perlu dua array linear, INFO[K] yang berisi bagian informasi dan LINK[K] yang berisi nextpointer field dari LIST. Seperti dijelaskan di atas, LIST memerlukan suatu nama variabel, yaitu START yang berisi lokasi awal dari list. Karena subscript array INFO dan LINK biasanya positif, kita akan menggunakan NULL = 0.

Contoh linked list berikut ini menandakan bahwa node suatu list tidak harus menempati elemen yang berdekatan dalam array INFO dan LINK, dan boleh terdapat satu atau lebih list dalam array INFO dan LINK yang sama. Tetapi, tiap list harus mempunyai variable pointer sendiri yang menunjukkan lokasi node pertamanya.



Gambar 7. 3. Contoh Linked List

Contoh:

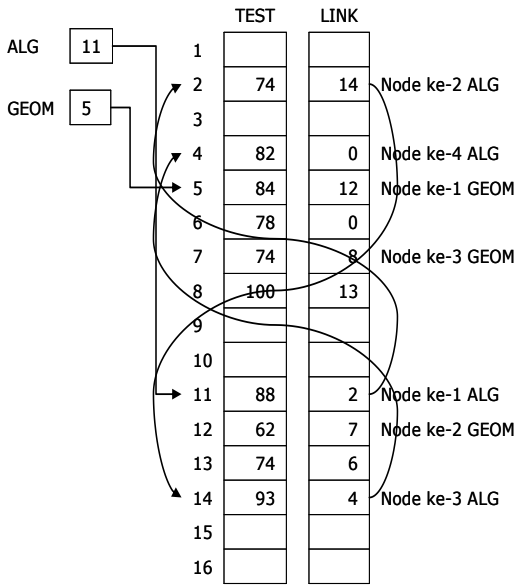
Gambar di atas menggambarkan linked list dalam memori, dimana tiap node mengandung satu karakter. Kita akan memperoleh suatu daftar karakter atau suatu string sebagai berikut:

START = 9, jadi INFO[9] = N, karakter pertama.
LINK[9] = 3, jadi INFO[3] = O, karakter kedua.
LINK[3] = 6, jadi INFO[6] = □ (blank), karakter ketiga.
LINK[6] = 11, jadi INFO[11] = E, karakter keempat.
LINK[11] = 7, jadi INFO[7] = X, karakter kelima.
LINK[7] = 10, jadi INFO[10] = I, karakter keenam.
LINK[10] = 4, jadi INFO[4] = T, karakter ketujuh.
LINK[4] = 0, nilai NULL menandakan akhir dari list

Contoh:

Gambar di bawah melukiskan bagaimana dua list yang berisi nilai test, ALG dan GEOM, dimana node keduanya ada pada array linear TEST dan LINK yang sama. Perhatikan nama list juga digunakan sebagai nama variabel pointer. ALG dan GEOM masing-masing berisi

nilai 11 dan 5, lokasi node pertamanya. Kalau kita ikuti pointer, kita dapat melihat ALG dan berisi nilai test 88, 74, 93, 82, dan GEOM berisi nilai test 84, 62, 74, 100, 74, 78.

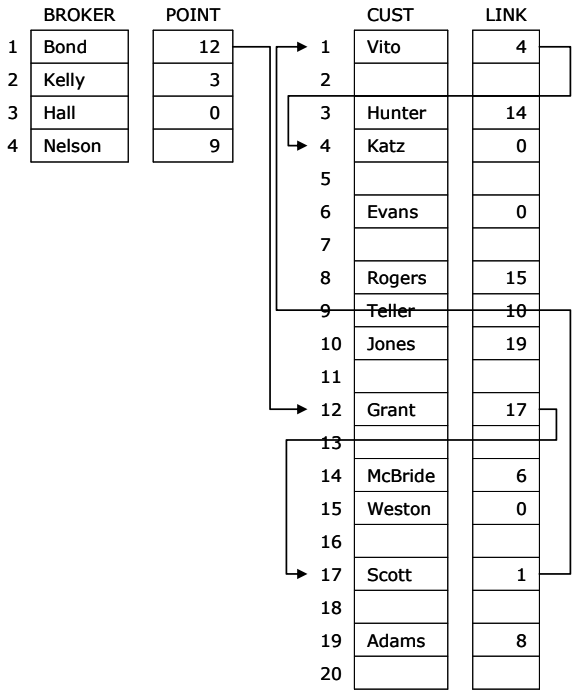


Gambar 7. 4. ALG dan GEOM

Contoh:

Suatu perusahaan pialang mempunyai 4 orang broker dan tiap broker mempunyai daftar pelanggan tersendiri. Data dapat diorganisasikan seperti pada gambar di bawah. Seluruh daftar pelanggan ada pada array CUSTOMER yang sama, dan array LINK berisi nextpointer field node dari list. Juga terdapat array BROKER yang berisi daftar nama broker dan array pointer POINT, dimana POINT[K] menunjuk ke awal daftar customer dari BROKER[K].

Maka, dapat dilihat pada gambar bahwa daftar customer Bond adalah Grant, Scott, Vito, Katz. Dengan cara yang sama, Kelly mempunyai customer: Hunter, McBride, Evans. Nelson mempunyai customer: Teller, Jones, Adams, Rogers dan Weston. Sedangkan Hall tidak mempunyai customer, karena POINT[3] = 0.



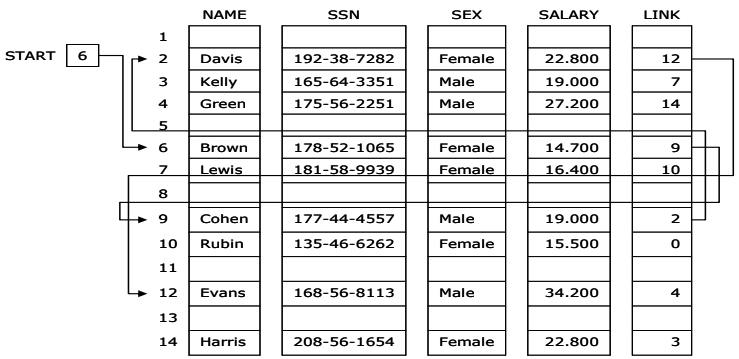
Gambar 7. 5. Contoh Linked List

Seperti dikatakan di awal, bagian informasi mungkin berisi suatu record dengan satu atau lebih item data. Dalam hal ini data harus disimpan dalam beberapa tipe struktur record atau dalam kumpulan array paralel, seperti yang diilustrasikan pada contoh berikut:

Contoh:

Misalkan file pegawai suatu perusahaan kecil berisi informasi untuk sembilan karyawannya dengan data sebagai berikut: Nama, No. KTP, Jenis Kelamin, dan Gaji. Biasanya, empat array parallel NAME, SSN, SEX dan SALARY diperlukan untuk menyimpan data tersebut. Gambar di bawah menunjukkan bagaimana data disimpan dalam linked list yang terurut secara alphabet menggunakan satu array tambahan LINK untuk daftar nextpointer field dan variabel START yang

menunjukkan record pertama dalam list. Perhatikan 0 digunakan sebagai null pointer.

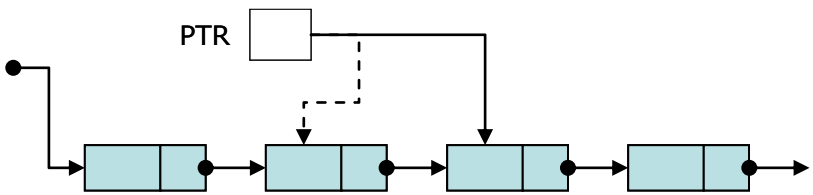


Gambar 7. 6. Contoh Linked List

7.2. Menjelajahi Linked List

Misalkan linked list LIST ada di memory dan disimpan dalam array linear INFO dan LINK dengan START yang menunjuk elemen pertama dan NULL menunjukkan akhir dari list. Misalkan kita ingin menjelajahi LIST untuk memproses tiap node. Bagian ini menyajikan suatu algoritma untuk menjelajahi list kemudian menggunakannya pada beberapa aplikasi.

Algoritma traversing ini akan menggunakan variable pointer PTR yang akan menunjukkan node yang sedang diproses. Selain itu LINK[PTR] menunjuk ke node berikutnya yang akan diproses. Pernyataan PTR:= LINK[PTR] memindahkan penunjuk ke node berikutnya dalam list.



Gambar 7. 7. Alur List

Detail algoritma akan dijelaskan berikut ini. Pertama, PTR diinisialisasikan ke START. Kemudian memproses INFO[PTR] informasi pada node pertama. Update PTR menggunakan pernyataan PTR:= LINK[PTR], sehingga PTR menunjuk ke node yang kedua. Kemudian memproses INFO[PTR], informasi pada node kedua. Sekali lagi update PTR menggunakan pernyataan PTR:= LINK[PTR], dan kemudian memproses INFO[PTR], informasi pada node ketiga dan seterusnya sampai PTR = NULL yang menandakan akhir dari list.

```
Algoritma (Menjelajahi Linked List) Misalkan
          LIST adalah suatu linked list dalam
          memori. Algoritma ini akan menjelajahi
          LIST, melakukan operasi PROCESS untuk
          tiap elemen dalam LIST. Variabel PTR
          digunakan untuk menunjuk node yang
          sedang diproses.

          1. Set PTR:= START
          2. Repeat Step 3 dan 4 While PTR ≠ NULL
          3.     Apply PROCESS to INFO[PTR]
          4.     Set PTR:= LINK[PTR]
             [End of Step 2 loop]
          5. Exit.
```

Contoh:

Procedure berikut ini mencetak informasi tiap node dalam linked list. Karena prosedur harus menjelajahi list, maka prosedur terlihat sangat mirip dengan.

```
Procedure PRINT(INFO, LINK, START)
          Procedure ini mencetak informasi tiap
          node dalam list.
```

```
1. Set PTR:= START
2. Repeat Step 3 dan 4 While PTR ≠ NULL
3.     Write: INFO[PTR]
4.     Set PTR:= LINK[PTR]
   [End of Step 2 loop]
5. Return.
```

Contoh:

Procedure berikut menghitung NUM jumlah elemen dalam linked list.

```
Procedure COUNT(INFO, LINK, START, NUM)

1. Set NUM:= 0.
2. Set PTR:= START
3. Repeat Step 4 dan 5 While PTR ≠ NULL
4.     Set NUM:= NUM + 1
5.     Set PTR:= LINK[PTR]
   [End of Step 2 loop]
6. Return.
```

Perhatikan bahwa procedure menjelajahi list untuk menghitung jumlah elemen, karena prosedur sangat mirip dengan **Error! Reference source not found.**. Kita memerlukan langkah inisialisasi untuk variabel NUM sebelum menjelajahi list. Procedure di atas dapat ditulis seperti berikut ini:

```
Procedure COUNT(INFO, LINK, START, NUM)

1. Set NUM:= 0.
2. Call  Error! Reference source not found.,
   replacing the processing step by:
```

```
                    Set NUM:= NUM + 1.
    Return.
```

7.3. Pencarian dalam Linked List

Misalkan LIST adalah linked list yang ada di memori. Misalkan suatu informasi ITEM diberikan. Bagian ini membahas dua algoritma pencarian untuk mencari lokasi node LOC dimana ITEM pertama kali muncul di LIST. Algoritma pertama mengasumsikan data dalam LIST tidak diurutkan, sedangkan algoritma kedua mengasumsikan data telah terurut.

Jika ITEM adalah suatu nilai kunci dan kita mencari ke seluruh file yang mengandung ITEM maka ITEM hanya akan muncul satu kali dalam LIST.

LIST tidak Terurut

Misalkan data dalam LIST tidak diurutkan. Maka kita mencari ITEM dalam LIST dengan cara menjelajahi list menggunakan variable pointer PTR dan membandingkan ITEM dengan isi dari INFO[PTR] tiap node, satu per satu. Sebelum kita memperbaharui nilai PTR dengan PTR:= LINK[PTR], kita harus melakukan dua pengujian. Pertama kita harus menguji untuk melihat apakah kita sudah ada di akhir list dalam hal ini kita melihat apakah PTR = NULL, jika tidak, kemudian kita ujia apakah INFO[PTR] = ITEM. Kedua buah test tersebut tidak dapat dilakukan bersamaan, karena INFO[PTR] tidak didefinisikan saat PTR = NULL. Maka, kita gunakan test pertama untuk mengontrol eksekusi perulangan, dan kita letakkan test kedua di bagian dalam perulangan.

```
Algoritma SEARCH(INFO, LINK, START, ITEM, LOC)
          LIST adalah linked list di memori.
          Algoritma mencari lokasi node LOC
          dimana ITEM pertama kali ada dalam list
          atau menentukan LOC = NULL.
```

```
1. Set PTR:= START
2. Repeat step 3 While PTR ≠ NULL
3.      If ITEM = INFO[PTR] then:
                Set LOC:= PTR and Exit.
        Else:
                Set PTR:= LINK[PTR].
        [End If Structure]
    [End of Step 2 Loop]
4. Set LOC:= NULL
5. Exit.
```

Kompleksitas algoritma ini sama dengan algoritma linear search untuk array linear pada bab 4. *Worst case* waktu yang dibutuhkan untuk menjalankan algoritma proporsional terhadap jumlah n elemen dalam list, dan average case waktu yang dibutuhkan untuk menjalankan algoritma kira-kira sama dengan n/2.

Contoh:

Perhatikan file karyawan pada gambar di atas. Modul berikut membaca no KTP NNN dari seorang karyawan dan memberikan kenaikan gaji 5% untuk karyawan tersebut.

```
Modul
          1. Read: NNN
          2. Call SEARCH(SSN, LINK, START, NNN,
             LOC)
          3. If LOC ≠ NULL, Then:
                Set SALARY[LOC]:= SALARY[LOC] +
          0.05 * SALARY[LOC]
            Else:
                Write: NNN is not in file
             [End of If Structure]
```

```
        4. Return.
```

List Terurut

Misalkan data dalam LIST telah dalam keadaan terurut. Kita akan mencari ITEM dalam LIST dengan cara menjelajahi list menggunakan variabel pointer PTR dan membandingkan ITEM dengan isi dari INFO[PTR] tiap node, satu persatu.

```
Algoritma  SRCHSL(INFO, LINK, START, ITEM, LOC)
          LIST adalah list terurut dalam memori.
          Algoritma ini akan mencari lokasi LOC
          dari node dimana ITEM pertama kali
          ditemukan dalam LIST atau menyatakan
          LOC = NULL.

          1. Set PTR:= START.
          2. Repeat Step 3 While PTR ≠ NULL
          3.     If ITEM < INFO[PTR], then:
                       Set PTR:= LINK[PTR].
                 Else If ITEM = INFO[PTR], then:
                       Set LOC:= PTR, and Exit.
                 Else:
                       Set LOC:= NULL, and Exit.
                 [End If Structures]
            [End of Step 2 Loop].
          4. Set LOC:= NULL.
          5. Exit.
```

Ingat dengan array linear yang terurut kita dapat menggunakan binary search dimana running timenya proporsional terhadap $\log_2 n$. Di lain sisi, algoritma binary search tidak bisa diterapkan pada suatu linked list terurut, karena pada link list tidak terdapat index yang menentukan

elemen tengah dari list. Hal ini merupakan salah satu kerugian penggunaan linked list sebagai struktur data.

Contoh:

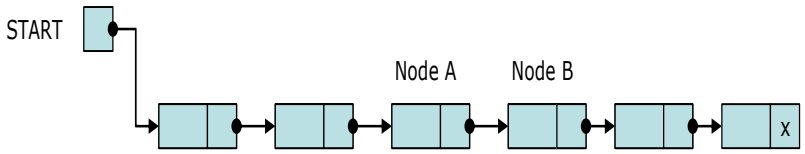
Perhatikan file karyawan pada gambar di atas, module berikut membaca nama EMP dari karyawan tertentu dan memberikan kenaikan 5% gajinya. Bandingkan dengan contoh di atas.

```
1. Read: EMPNAME.
2. Call  SRCHSL(NAME,  LINK,  START,  EMPNAME,
   LOC)
3. If LOC ≠ NULL, then:
         Set SALARY[LOC]:= SALARY[LOC] + 0.005
         * SALARY[LOC].
   Else:
         Write: EMPNAME is not in list.
   [End of IF Structure]
4. Return.
```

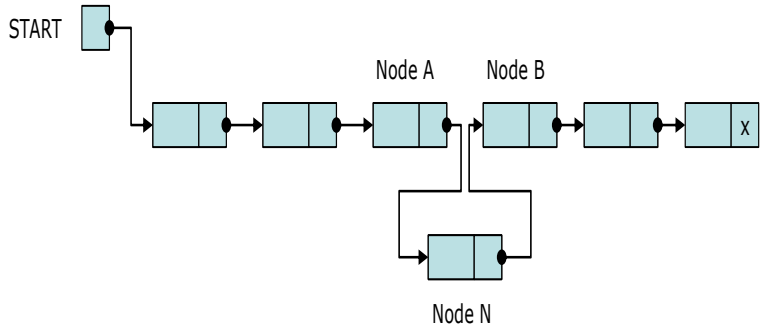
Perhatikan, sekarang kita dapat menggunakan algoritma pencarian, karena list sudah terurut secara alfabet.

7.4. Penyisipan ke dalam Linked List

Misalkan LIST adalah suatu linked list yang mempunyai node berurutan A dan B, sperti pada **Error! Reference source not found.**a. Misalkan node N disisipkan ke dalam list antara node A dan B. Diagram skematik penyisipan dapat dilihat pada **Error! Reference source not found.**b. Perhatikan, node A sekarang menunjuk ke node baru N dan node N menunjuk ke node B, yang sebelumnya ditunjuk oleh A.



(a) Sebelum penyisipan



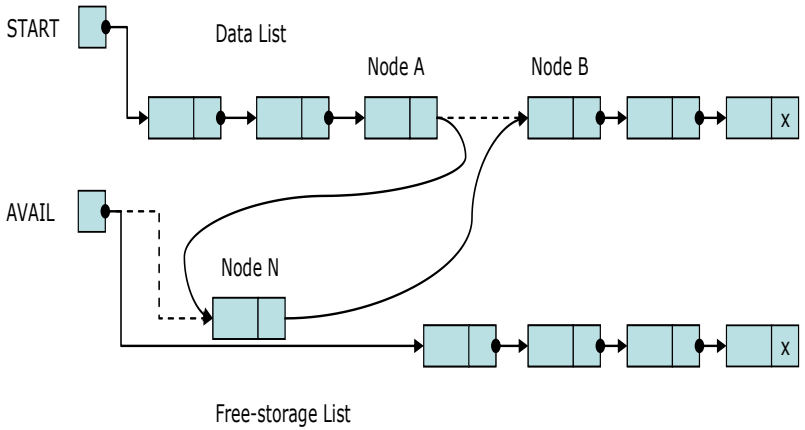
(b) Sesudah penyisipan

Gambar 7. 8. Penyisipan ke dalam Linked List

Misalkan linked list tersebut dijaga di memori dalam bentuk LIST(INFO, LINK, START, AVAIL), **Error! Reference source not found.** tidak memperhitungkan ruang memori untuk node baru N berasal dari list AVAIL. Secara khusus, untuk kemudahan pemrosesan, node pertama dalam list AVAIL akan digunakan sebagai node baru N. Diagram skematik penyisipan seperti di atas dapat dilihat pada **Error! Reference source not found.**. Perhatikan bahwa tiga pointer field diubah sebagai berikut:

1. Field nextpointer dari node A sekarang menunjuk ke node baru N, yang sebelumnya ditunjuk oleh AVAIL.
2. AVAIL sekarang menunjuk ke node kedua dalam *free-storage list*, yang sebelumnya ditunjuk oleh node N.
3. Field nextpointer dari node N sekarang menunjuk ke node B, yang sebelumnya ditunjuk oleh A.

Terdapat dua kasus khusus. Jika node baru N adalah node pertama dalam list, maka START akan menunjuk ke N dan jika node baru N adalah node terakhir dalam list, maka N akan berisi null pointer.



Gambar 7. 9. Contoh Penyisipan

Contoh:

1. Perhatikan gambar di atas, daftar terurut secara alfabet pasien dalam suatu rumah sakit. Misalkan seorang pasien Hughes dimasukkan ke rumah sakit. Perhatikan bahwa:
    a. Huges akan diletakkan di bed no 10, bed pertama yang tersedia.
    b. Hugges akan disisipkan di list antara Green dan Kirk.

    Terjadi tiga perubahan dalam field pointer sebagai berikut:

    a. a. Link[8] = 10. Sekarang Green menunjuk ke Hughes.
    b. Link[10] = 1. Sekarang Hughes menunjuk ke Kirk.
    c. AVAIL = 2. Sekarang AVAIL menunjuk ke bed yang tersedia berikutnya.

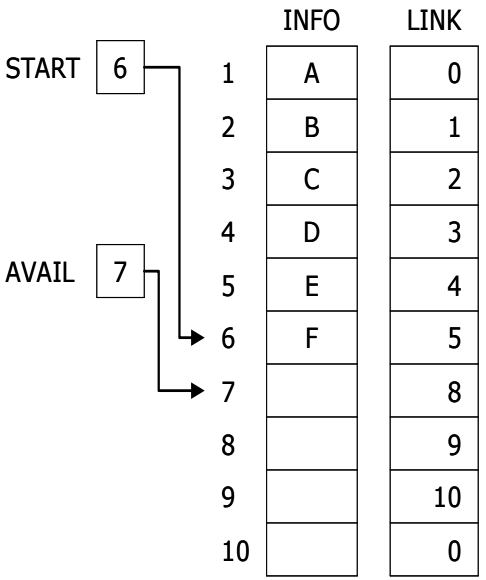
2. Perhatikan gambar di bawah, daftar broker dan customernya. Karena daftar customer tidak terurut, kita akan asumsikan bahwa tiap customer baru ditambahkan ke awal list. Misalkan Gordan adalah customer baru Kelly. Perhatikan bahwa:

    a. Gordan diletakkan ke CUSTOMER[11], node pertama yang tersedia.
    b. Godan disisipkan sebelum Hunter, sebelum customer pertama Kelly.

    Terdapat tiga perubahan dalam field pointer sebagai berikut:

    a. POINT[2] = 11. Sekarang list diawali oleh Gordon.
    b. LINK[11] = 3. Sekarang Gordon menunjuk ke Hunter.
    c. AVAIL = 18. Sekarang AVAILL menunjuk ke node yang tersedia berikutnya.

3. Misalkan elemen data A, B, C, D, E dan F disisipkan ke dalam list kosong, lihat gambar di bawah. Kita asumsikan tiap node baru disisipkan di awal list. Maka, setelah keenam elemen tersebut dimasukkan, F akan menunjuk ke E, E akan menunjuk ke D dan seterusnya sampai dengan A yang berisi null pointer. Juga, AVAIL = 7, node pertama yang tersedia setelah enam penyisipan dan START = 6, lokasi node pertama F. Gambar di bawahmenunjukkan daftar baru dimana n = 10.



Gambar 7. 10. Contoh Penyisipan

Algoritma Penyisipan

Algoritma yang menyisipkan node ke dalam linked list mempunyai berbagai situasi. Berikut akan dibahas tiga diantaranya. Pertama, menyisipkan node ke awal list, kedua, menyisipkan node setelah node tertentu, dan ketiga menyisipkan node ke dalam list terurut. Algoritma berikut kita asumsikan bahwa linked list di memori dalam bentuk LIST(INFO, LINK, START, AVAIL) dan variabel ITEM mengandung informasi baru yang akan ditambahkan ke dalam list.

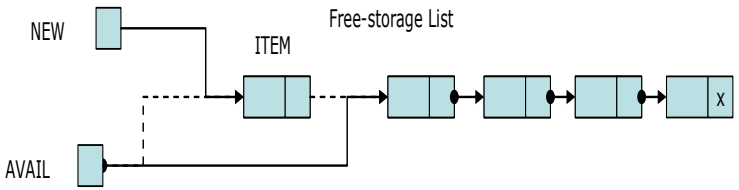
Karena algoritma penyisipan ini akan menggunakan sebuah node dalam list AVAIL, seluruh algoritma berikut akan menyertakan langkah-langkah sebagai berikut:

1. Memeriksa apakah tersedia ruang dalam list AVAIL. Jika AVAIL = NULL, maka algoritma akan mencetak pesan OVERFLOW.
2. Memindahkan node pertama dari list AVAIL. Menggunakan variabel NEW untuk melacak lokasi node baru, langkah ini dapat diimplementasikan menggunakan dua pernyatan berikut:

NEW:= AVAIL, AVAIL:= LINK[AVAIL]

3. Mencopy informasi baru ke dalam node baru, dengan kata lain INFO[NEW]:= ITEM.

Diagram skematik dari dua langkah terakhir di atas dapat dilihat pada gambar di bawah.



Gambar 7. 11. Diagram Skematik

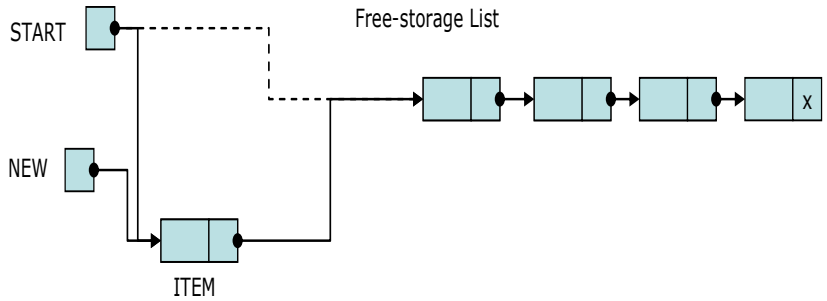
Penyisipan pada Awal List

Misalkan suatu linked list tidak perlu diurutkan dan tidak ada alasan untuk menyisipkan node baru ke tempat tertentu dalam list. Maka tempat paling mudah untuk menyisipkan node adalah di bagian awal list.

```
Algoritma INSFIRST(INFO, LINK, START, AVAIL,
ITEM)
          Algoritma ini menyisipkan ITEM
sebagai node pertama dalam list.

          1. If AVAIL = NULL, then Write:
             OVERFLOW and Exit.
          2. Set NEW:= AVAIL and AVAIL:=
             LINK[AVAIL]
          3. Set INFO[NEW]:= ITEM
          4. Set LINK[NEW]:= START
          5. Set START:= NEW
          6. Exit.
```

Langkah 1 sampai 3 telah dibahas sebelumnya, diagram skematik langkah 2 dan 3 dapat dilihat pada gambar **Error! Reference source not found.**. Berikut adalah diagram skematik langkah ke 4 dan ke 5.



Gambar 7. 12. Penyisipan Pada Awal List

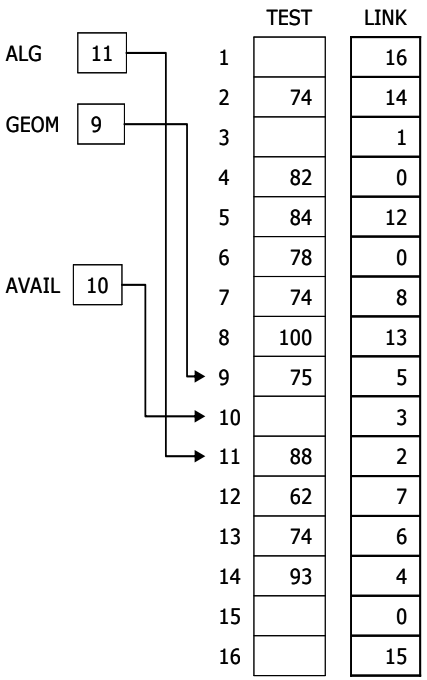
Contoh:

Perhatikan daftar test pada gambar di bawah. Misalkan nilai test 75 akan ditambahkan pada awal list geometri. Kita akan mensimulasikan algoritma. Perhatikan ITEM = 75, INFO = TEST dan START = GEOM.

INSFIRST(TEST, LINK, GEOM, AVAIL, ITEM)

1. Karena AVAIL ≠ NULL, control akan diteruskan ke langkah kedua.
2. NEW = 9, maka AVAIL = LINK[9] = 10.
3. TEST[9] = 75.
4. LINK[9] = 5.
5. GEOM = 9
6. Exit.

Gambar di bawah menunjukkan struktur data setelah 75 ditambahkan pada list geometri. Perhatikan, hanya tiga pointer yang berbubah yaitu AVAIL, GEOM dan LINK[9].



Gambar 7. 13. Penyisipan Pada Awal List

Penyisipan Setelah Node Tertentu

Misalkan kita diberi nilai tertentu dari LOC dimana LOC adalah lokasi dari node A dalam LIST atau LOC = NULL. Berikut ini adalah satu algoritma yang menyisipkan ITEM ke dalam LIST sehingga ITEM ada setelah node A, jika LOC = NULL, maka ITEM adalaah node pertama.

Misalkan N dinyatakan sebagai node baru (dimana lokasinya adalah NEW). Jika LOC = NULL, dan N disisipkan sebagai node pertama dalam LIS seperti pada **Error! Reference source not found.**. Dilain sisi, seperti yang ditunjukkan pada **Error! Reference source not found.**, kita misalkan node N menunjuk node B (pada awalnya diikuti oleh node A) menggunakan penyataan LINK[NEW]:= LINK[LOC] dan misalkan node A menunjuk ke node baru N menggunakan pernyataan LINK[LOC]:= NEW.

```
Algoritma INSLOC(INFO, LINK, START, AVAIL, LOC,
          ITEM)
          Algoritma ini menyisipkan ITEM
          sehingga ITEM berada pada lokasi
          sesudah lokasi LOC atau menyisipkan
          ITEM sebagai node pertama jika LOC =
          NULL.

          1. If AVAIL = NULL then Write:
             Overflow, and Exit.
          2. Set NEW:= AVAIL and AVAIL:=
             LINK[AVAIL]
          3. Set INFO [NEW]:= ITEM
          4. If LOC = NULL then:
                 Set LINK[NEW]:= START and
          START:= NEW
             Else:
                 Set LINK[NEW]:= LINK[LOC] and
          LINK[LOC]:= NEW.
             [End of If Structure]
          5. Exit.
```

Penyisipan pada Linked List Terurut

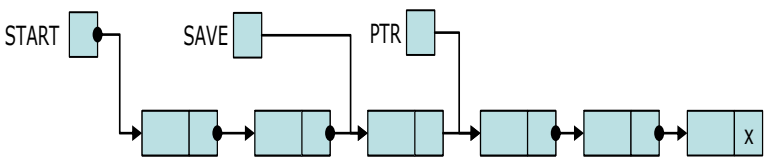
Misalkan ITEM akan disisipkan ke linked list terurut LIST. Maka ITEM harus disisipkan diantara node A dan B sehingga INFO(A) < ITEM ≤ INFO(B). Berikyt ini adalah procedure untuk mencari lokasi LOC dari node A, yaitu lokasi dimana nilainya kurang dari ITEM.

Menjelajahi list menggunakan variabel pointer PTR dan membandingkan ITEM dengan INFO[PTR] tiap node. Selama menjelajahi, simpan lokasi node sebelumnya menggunakan variable pointer SAVE, seperti pada **Error! Reference source not found.**. SAVE dan PTR akan diupdate menggunakan pernyataan SAVE:= PTR dan PTR:= LINK[PTR]. Penjelajahan dilanjutkan selama INFO[PTR]

> ITEM, atau dengan kata lain penjelajahan berhenti saat ITEM ≤ INFO[PTR]. Kemudian PTR menunjuk ke node B dan SAVE akan berisi lokasi dari node A.

```
Procedure FIND(INFO, LINK, START, ITEM, LOC)
          Prosedur ini mencari lokasi LOC node
          terakhir dalam list terurut sehingga
          INFO[LOC] < ITEM atau menyatakan LOC =
          NULL.

          1. If START = NULL then, Set LOC:=
             NULL, and Return
          2. If ITEM < INFO[START], then Set
             LOC:= NULL and Return
          3. Set SAVE:= START and PTR:=
             LINK[START]
          4. Repeat Step 5 and 6 while PTR ≠ NULL
          5.     If ITEM < INFO[PTR], then:
                       Set LOC:= SAVE, and Return
                 [End of if Structure]
          6.     Set SAVE:= PTR and PTR:=
             LINK[PTR]
             [End of Step 4 Loop]
          7. Set LOC:= SAVE
          8. Return.
```



Gambar 7. 14. Penyisipan pada Linked List Terurut

Sekarang kita sudah mempunyai komponen untuk menjalankan algoritma yang akan menyisipkan ITEM ke dalam linked list. Berikut adalah algoritma menggunakan dua procedure terdahulu.

```
Algoritma INSSRT(INFO, LINK, START, AVAIL, ITEM)
          Algoritma ini menyisipkan ITEM ke
          dalam linked list terurut.

          1. Call FINDA(INFO, LINK, START, ITEM,
             LOC)
          2. Call INSLOC(IN FO, LINK, START,
             AVAIL, LOC, ITEM)
          3. Exit.
```

Contoh:

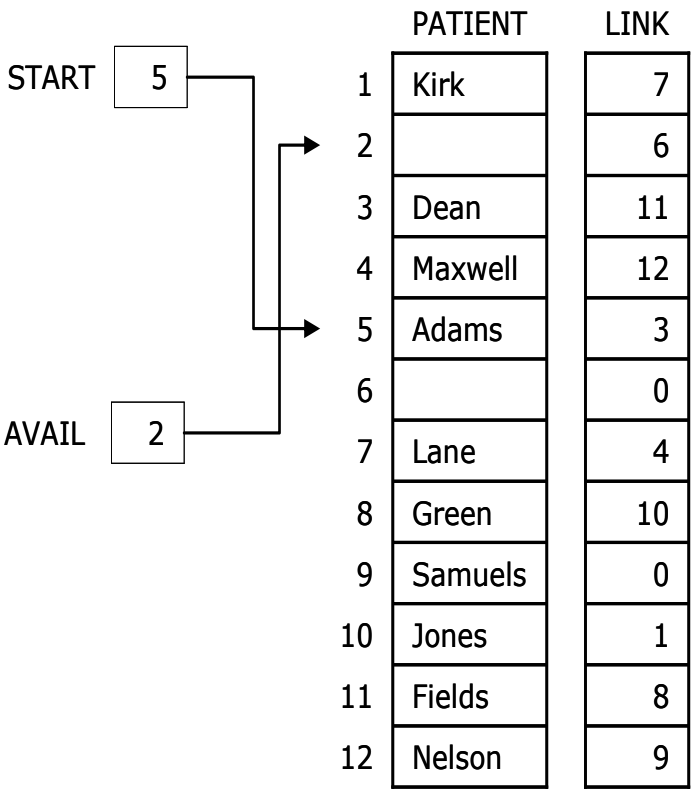
Perhatikan daftar pasien pada di atas. Misalkan Jones akan ditambahkan ke dalam list pasien. Kita akan mensimulasikan algoritma atau lebih jelasnya, kita akan mensimulasika algoritma berikut perhatikan ITEM = Jones dan INFO = BED.

a. FINDA(BED, LINK, START, ITEM, LOC)
   1. Karena START ≠ NULL, control akan dipindahkan ke Step 2.
   2. Karena BED[5] = Adams < Jones, control dipindahkan ke Step 3.
   3. SAVE = 5 dan PTR = LINK[5] = 3
   4. Langkah 5 dan 6 diulang sebagai berikut:
      - BED[3] = Dean < Jones, jadi SAVE = 3 dan PTR = LINK[3] = 11.
      - BED[11] = Fields < Jones, jadi SAVE = 11 dan PTR = LINK[11] = 8.
      - BED[8] = Green < Jones, jadi SAVE = 8 dan PTR = LINK[8] = 1.

- Karena BED[1] = Kirk > Jones, kita dapat LOC = SAVE = 8 dan return.

b. INSLOC(BED, LINK, START, AVAIL, LOC, ITEM), sekarang LOC = 8.
   1. Karena AVAIL ≠ NULL, control dipindahkan ke Step 2.
   2. NEW = 10 dan AVAIL = LINK[10] = 2.
   3. BED[10] = Jones.
   4. Karena LOC ≠ NULL, kita temukan bahwa: LINK[10] = LINK[8] = 1 dan LINK[8] = NEW = 10.
   5. Exit.

Error! Reference source not found. menunjukkan struktur data setelah Jones ditambahkan ke dalam list pasien. Perlu ditekankan disini adalah hanya tiga pointer yang akan diubah yaitu AVAIL, LINK[10] dan Link[8].



Gambar 7. 15. Penyisipan pada Linked List Terurut

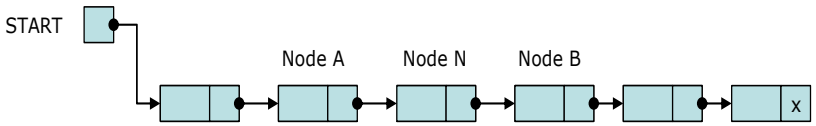
Penggandaan

Misalkan kita ingin menggandakan semua atau sebagian list tertentu, atau misalkan kita ingin membuat list baru yang merupakan gabungan dua list tertentu. Hal tersebut dapat dilakukan dengan mendefinisikan list kosong dan kemudian menambahkan elemen yang bersangkutan ke dalam list, satu per satu dengan menggunakan algoritma penyisipan. List kosong didefinisikan dengan memilih nama variabel atau pointer untuk list, seperti NAMA kemudian menentukan NAMA:= NULL.

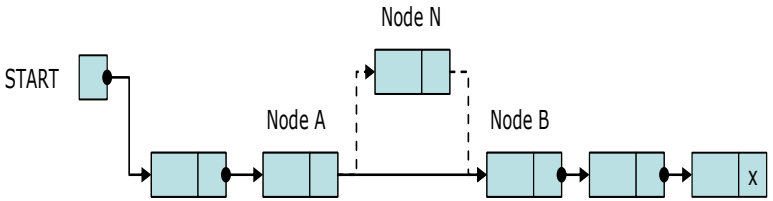
7.5. Penghapusan dari Linked List

Misalkan LIST adalah suatu linked list dengan node N antara node A dn B seperti pada **Error! Reference source not found.**a. Misalkan node N akan dihapus dari linked list. Diagram skematik penghapusan dapat dilihat pada **Error! Reference source not found.**b. Penghapusan muncul segera setelah field nextpointer dari node A berubah sehingga menunjuk ke node B. (Sejalan dengan hal itu, ketika kita melakukan penghapusan, kita harus menyimpan alamat dari node yang bersebelahan sebelum node yang akan dihapus).

Misalkan linked list dikelola di memori dengan bentuk LIST(INFO, LINK, START, AVAIL).



(a) sebelum penghapusan



(b) Setelah Penghapusan

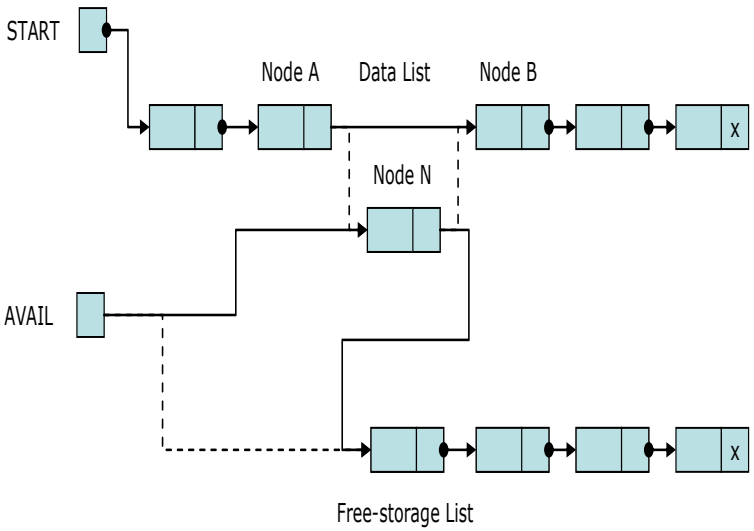
Gambar 7. 16. Penghapusan dari Linked List

Ketika node N dihapus dari list, kita akan segera mengembalikan ruang memori ke list AVAIL. Secara khusus, untuk memudahkan pemrosesan, node yang dihapus akan dikembalikan ke awal list AVAIL. Untuk lebih jelasnya dapat dilihat pada diagram skematik **Error! Reference source not found.**. Perhatikan perubahan tiga pointer di bawah ini:

a. Nextpointer node A sekarang menunjuk ke node B, dimana sebelumnya ditunjuk oleh node N.

b. Nextpointer node N sekarang menunjuk ke node awal dalam free pool, dimana sebelumnya ditunjuk oleh AVAIL.
c. AVAIL sekarang menunjuk ke node N yang telah dihapus.

Terdapat dua kasus khusus. Jika node N yang dihapus adalah node awal dalam list, maka START akan menunjuk ke node B, dan jika node N yang dihapus ada pada akhir list, maka node A akan berisi NULL pointer.



Gambar 7. 17. Penghapusan dari Linked List

Contoh:

a. Perhatikan daftar pasien di rumah sakit. Misalkan Green dikeluarkan, sehingga BED[8] sekarang kosong. Maka untuk menjaga urutan dalam linked list, berikut adalah tiga perubahan field pointer yang harus dilakukan: LINK[11] = 10, LINK[8] = 2 dan AVAIL = 8.

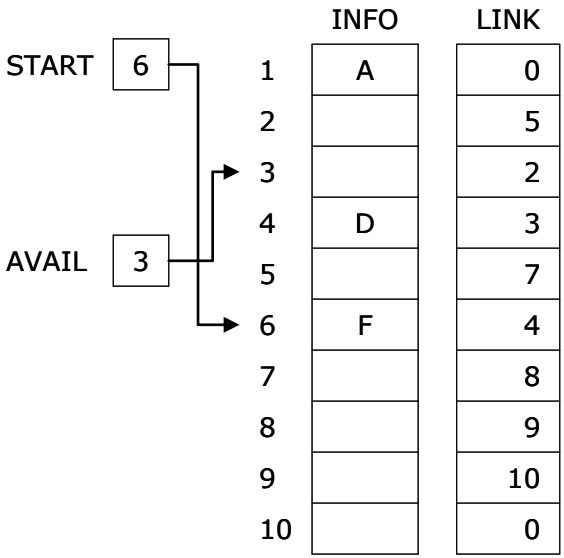
   Pada perubahan pertama, Fields yang pada awalnya berada sebelum Green, sekarang menunjuk ke Jones yang awalnya ada sesudah Green. Perubahan kedua dan ketiga menambahkan bed yang kosong ke list AVAIL. Perlu ditekankan bahwa sebelum

penghapusan, kita harus mencari node BED[11], yang awalnya ditunjuk oleh node BED[8].

b. Perhatikan daftar broker dan pelanggannya. Misalkan Teller, pelanggan pertama Nelson dihapus dari list pelanggan. Maka untuk menjaga urutan dalam linked list, berikut adalah tiga perubahan pointer yang harus dilakukan: POINT[4] = 10, LINK[9] = 11 dan AVAIL = 9.

   Pada perubahan pertama, Nelson sekarang menunjuk ke customer kedua yaitu Jones. Perubahan kedua dan ketiga menambahkan node kosong ke dalam list AVAIL.

c. Misalkan elemen data E, B dan C dihapus, secara berturut-turut dari list. List baru akan ditunjukkan pada gambar di bawah. Perhatikan bahwa terdapat tiga node yang tersedia yaitu: INFO[3], yang awalnya berisi C, INFO[2], yang awalnya berisi B dan INFO[5], yang awalnya berisi E. Perhatikan bahwa urutan node dalam list AVAIL terbalik.

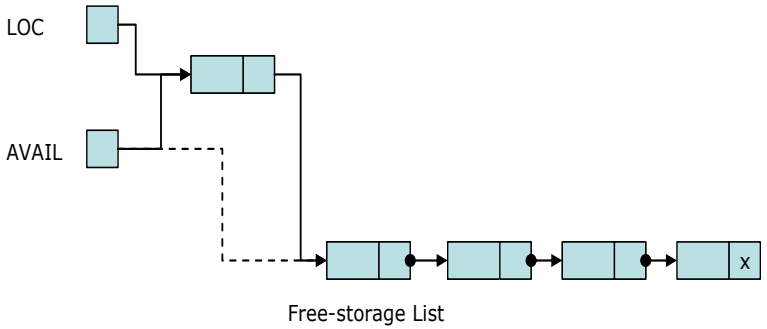


Gambar 7. 18. List Baru

Algoritma Penghapusan

Algoritma yang menghapus node dari link list ada dalam beberapa situasi. Berikut akan dibahas dua diantaranya. Pertama, penghapusan dilakukan berdasarkan node tertentu. Kedua, penghapusan dilakukan berdasarkan ITEM informasi yang diberikan. Semua algoritma mengasumsikan bahwa linked list ada di memori dalam bentuk LIST(INFO, LINK, START, AVAIL).

Semua algoritma penghapusan akan mengembalikan ruang memori dari node N yang dihapus ke awal list AVAIL. Sejalan dengan itu, semua algoritma kita akan mengandung sepasang pernyataan, dimana LOC adalah lokasi dari node N yang dihapus: LINK[LOC]:= AVAIL dan AVAIL:= LOC.



Gambar 7. 19. Algoritma Penghapusan Linked List

Beberapa algoritma mungkin ingin menghapus node awal atau node akhir dari list. Suatu algoritma harus memeriksa apakah terdapat node atau tidak. Jika tidak, dalam hal ini START = NULL maka algoritma harus mencetak pesan UNDERFLOW.

Penghapusan Node Berdasarkan Node Tertentu

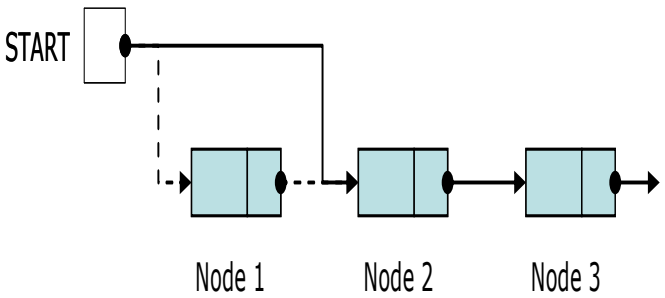
Misalkan LIST adalah suatu linked list dalam memori. Misalkan kita diberikan lokasi LOC dari node N dalam LIST. Lebih jauh, misalkan kita diberikan lokasi LOCP dari node sebelum N, atau jika N adalah node pertama, maka LOCP = NULL. Berikut adalah algoritma yang menghapus N dari list.

```
Algoritma DEL(INFO, LINK, START, AVAIL, LOC,
          LOCP)
          Algoritma ini akan menghapus node N
          dengan lokasi LOC. LOCP adalah lokasi
          node sebelum N atau jika N adalah node
          pertama, maka LOCP = NULL.

          1. If LOCP = NULL then:
                Set START:= LINK[START]
             Else:
                Set LINK[LOCP]:= LINK[LOC]
```

```
    [End of If Structure]
2. Set LINK[LOC]:= AVAIL and AVAIL:=
   LOC.
3. Exit.
```

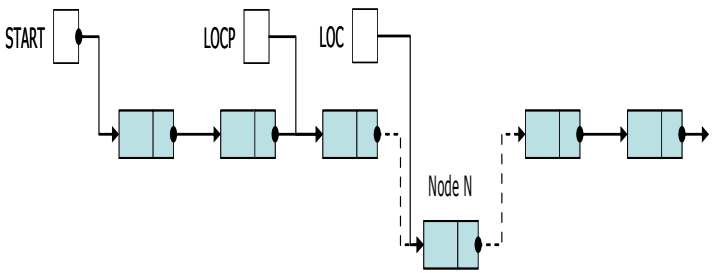
Diagram skematik dari pernyataan START:= LINK[START] yang secara efektif menghapus node pertama dari list.



Gambar 7. 20. Diagram Skematik

Diagram skematik dari pernyataan LINK[LOCP]:= LINK[LOC] yang secara efektif menghapus node N saat N bukan node pertama.

Kesederhanaan algoritma ini karena adanya fakta bahwa kita sudah mengetahui lokasi LOCP yaitu node sebelum node N. Dalam beberapa aplikasi, kita pertama harus mencari LOCP.



Gambar 7. 21. Algoritma LOCP

Menghapus Node Berdasarkan ITEM Informasi Tertentu

Misalkan LIST adalah suatu linked list dalam memory. Misalkan kita diberikan suatu ITEM informasi dan kita ingin menghapus node

pertama N yang mengandung ITEM dari LIST. (Jika ITEM adalah nilai kunci, maka hanya ada satu node yang mengandung ITEM). Ingat bahwa sebelum kita dapat menghapus N dari list, kita harus mengetahui lokasi node sebelum node N. Sejalan dengan itu, pertama kita akan membuat suatu prosedur yang mencari lokasi LOC dari node N yang mengandung ITEM dan lokasi LOCP node sebelum node N. Jika N adalah node pertama, maka kita tentukan LOCP = NULL, dan jika ITEM tidak ada dalam LICT, maka kita nyatakan LOC = NULL.

Menjelajahi list, menggunakan variable pointer PTR dan membandingkan ITEM dengan INFO[PTR] tiap node. Selama menjelajahi, simpan lokasi dari node sebelumnya menggunakan variabel pointer SAVE seperti pada **Error! Reference source not found.**. SAVE dan PTR akan diupdate menggunakan pernyataan: SAVE:= PTR dan PTR:= LINK[PTR]. Penjelajahan dilanjutkan selama INFO[PTR] ≠ ITEM atau dengan kata lain, penjelajahan berhenti segera setelah ITEM = INFO[PTR]. Maka PTR akan berisi lokasi LOC dari node N dan SAVE berisi lokasi LOCP dari node sebelum node N.

```
Procedure FINDB(INFO, LINK, START, ITEM, LOC,
          LOCP)
          Procedure ini mencari lokasi LOC dari
          node pertama N yang mengandung ITEM dan
          lokasi LOCP dari node sebelum node N.
          Jika ITEM tidak terdapat dalam list,
          maka procedure akan menset LOC = NULL,
          dan jika ITEM terdapat pada node
          pertama, maka procedure akan menset
          LOCP = NULL.

          1. If START = NULL, then:
                 Set LOC:= NULL and LOCP:= NULL,
          and Return.
             [End of If Structure]
```

```
2. If INFO[START] = ITEM, then:
       Set LC:= START and LOCP = NULL,
and Return.
   [End of if Structure]
3. Set   SAVE:=   START   and   PTR:=
   LINK[START].
4. Repeat Step 5 and 6 While PTR ≠ NULL.
5.     If INFO[PTR] = ITEM, Then:
            Set LOC:= PTR and LOCP:=
SAVE, and Return.
       [End of If Structure]
6.     Set   SAVE:=   PTR   and   PTR:=
   LINK[PTR]
   [End of Step 4 Loop]
7. Set LOC:= NULL.
8. Return.
```

Sekarang kita dapat dengan mudah membuat suatu algoritma untuk menghapus node pertama N dari suatu linked list yang mengandung informasi ITEM tertentu. Kesederhanaan dari algoritma ini adalah karena tugas untuk mencari lokasi N dan lokasi sebelum node tersebut sudah dilakukan pada algoritma berikut.

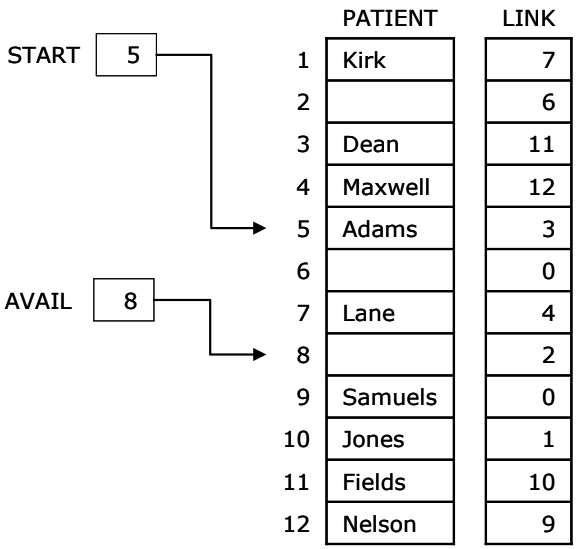
```
Algoritma DELETE(INFO, LINK, START, AVAIL, ITEM)
          Algoritma ini mengahpus node pertama N
          dalam linked list yang mengandung ITEM
          informasi tertentu.

          1. Call FINDB(INFO, LINK, START, ITEM,
             LOC, LOCP)
          2. If LOC = NULL, then: Write: ITEM not
             in list, and Exit.
          3. If LOCP = NULL, then:
                 Set START:= LINK[LOC].
```

```
        Else:
            Set LINK[LOCP]:= LINK[LOC]
        [End of If Structure]
    4. Set LINK[LOC]:= AVAIL and AVAIL:=
       LOC.
    5. Exit.
```

Contoh:

Perhatikan daftar pasien. Misalkan pasien Green dikeluarkan. Kita akan mensimulasikan algoritma untuk mencari lokasi LOC dari Green dan LOCP lokasi pasien sebelum Green. Kemudian kita simulasikan **Error! Reference source not found.** untuk menghapus Green dari list. Disini ITEM = Green, INFO = BED, START = 5 dan AVAIL = 2.



Gambar 7. 22. Penghapusan dari Linked List

a. FINB(BED, LINK, START, ITEM, LOC, LOCP)
    1. Karena Start ≠ NULL, control dipindahkan ke Step 2.
    2. Karena BED[5] = Adam ≠ Green, control dipindahkan ke Step 3.

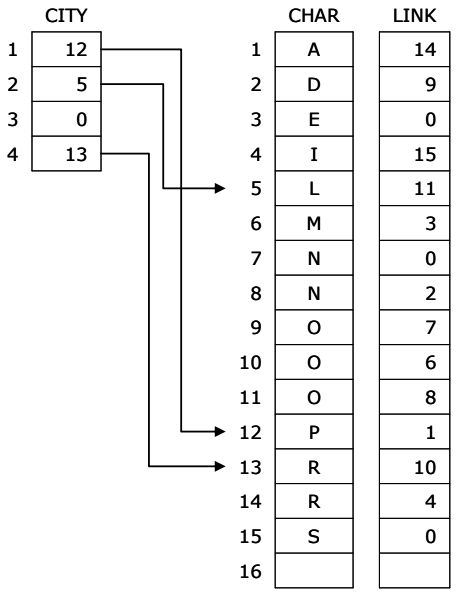
3. SAVE = 5 dan PTR = LINK[5] = 3.
4. Langkah 5 dan 6 diulang sebagai berikut:
   i. BED[3] = Dean ≠ Green, jadi SAVE = 3 dan PTR = LINK[3] = 11.
   ii. BED[11] = Fields ≠ Green, jadi SAVE = 11 dan PTR = LINK[11] = 8.
   iii. BED[8] = Green, jadi kita dapatkan LOC = PTR = 8 dan LOCP = SAVE = 11 dan Return.

b. DELLOC(BED, LINK, START, AVAIL, ITEM)
   1. Call FINDB(BED, LINK, START, ITEM, LOC, LOCP). LOC = 8 dan LOCP = 11.
   2. Karena LOC ≠ NULL, control dipindahkan ke Step 3.
   3. Karena LOCP ≠ NULL, kita dapat LINK[11] = LINK[8] = 10.
   4. LINK[8] = 2 dan AVAIL = 9/
   5. Exit.

Gambar di atas menunjukkan struktur data setelah Green dihapus dari daftar pasien. Perlu ditekankan di sini hanya terjadi tiga perubahan pointer yaitu LINK[11], LINK[8] dan AVAIL.

7.6. Soal Latihan

1. Tentukan empat string yang disimpan dalam linked list pada gambar berikut.



2. Berikut adalah nama yang akan dimasukkan ke dalam array INFO:

   Mary, June, Barbara, Paula, Diana, Audrey, Karen, Nancy, Ruth, Eileen, Sandra, Helen.
   Maka, INFO[1] = Mary, INFO[2]:= June, …, INFO[12] = Helen. Pasangkan nilai pada array LINK dan variable START sehingga INFO, LINK dan START membentuk daftar nama yang terurut secara alphabet.
3. Perhatikan list alfabet pasien pada gambar **Error! Reference source not found.**. Tentukan perubahan struktur data jika:
   a. Walters ditambahkan ke dalam list.
   b. Kirk dihapus dari list.

DAFTAR PUSTAKA

Ellzey, Roy S. 1991. *Data Structures for Computer Information Systems*. Edisi Kedua. Maxwell Macmillan International Edition.

Jogiyanto. *Struktur Data dengan Pascal*. Andi Offset. Yogyakarta.

Leung, Kruse dan Tondo. 1997. *Data Structures & Program Design in C*. Edisi Kedua. Prentice-Hall.

Lipschutz, Seymour. 1986. *Schaum's Outline of Data Structure*. Cetakan Pertama. Penerbit McGraw-Hill Book Company.

Sanjaya, Dwi. 2001. *Berpetualang dengan Struktur Data di Planet Pascal*. J & J Learning Yogyakarta.

Santosa, Insap. *Struktur Data Menggunakan Turbo Pascal*. Andi Offset. Yogyakarta.

Shaffer, Clifford A. 1997. *A Practical Introduction to Data Structure and Algorithm Analysis*. Prentice-Hall.

Struktur Data

Pada Logika & Algoritma Pemrograman

MUHAMMAD SYAHID PEBRIADI

Algoritma berasal dari kata Algoris dan Ritmis yang pertama kali diungkapkan oleh Abu Ja'far Mohammed Ibn Musa Al-Khowarismi pada tahun 825 M dalam bukunya yang berjudul Al-Jabr Wa-al Muqabla.

Dalam bidang pemrograman, algoritma didefinisikan sebagai suatu metode khusus yang tepat dan terdiri dari serangkaian langkah yang terstruktur dan dituliskan secara sistematis yang akan dikerjakan untuk menyelsesaikan masalah dengan bantuan komputer.

Algoritma merupakan pola pikir terstruktur yang berisi tahap-tahap penyelesaian masalah yang dapat disajikan dengan teknik tulisan maupun dengan gambar. Penyajian algoritma dalam bentuk tulisan menggunakan pseudo code (kode semu). Sedangkan penyajian algoritma dalam bentuk gambar menggunakan flowchart (diagram alir).

Penerbit Poliban Press
Redaksi :
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry, Pangeran, Komp. Kampus ULM, Banjarmasin Utara
Telp : (0511)3305052
Email : press@poliban.ac.id

ISBN 978-623-7694-85-4 (PDF)

ISBN 978-623-7694-84-7