BAB II
MATERI PENUNJANG

2.1. List

List adalah suatu koleksi dari simpul-simpul atau elemen-elemen informasi yang disusun dalam urutan tertentu.

Struktur data yang berkorespondensi itu harus dapat menentukan secara efisien letak simpul pertama dalam strukturnya, letak yang terakhirnya dan letak pendahulu (*predecessor*) dan pengikut (*successor*)-nya (jika ada) dari sebarang simpul yang diberikan.

Struktur data yang demikian sering direpresentasikan secara grafik dengan kotak dan panah. Informasi yang dilampirkan ke suatu simpul diperlihatkan di bagian dalam kotak yang berkoresponden dan panah menunjukkan transmisi dari suatu simpul ke pengikutnya.

2.2. Linked-List

Linked-list adalah suatu strukur data dengan objek-objeknya disusun dalam suatu urutan linier. Urutan dalam linked-list ditentukan oleh suatu pointer dalam tiap objek.

Implementasi suatu linked-list berhubungan dengan tiap simpul (*node*) yang ditunjuk oleh suatu pointer. Pointer yang belum menunjuk ke suatu simpul nilainya

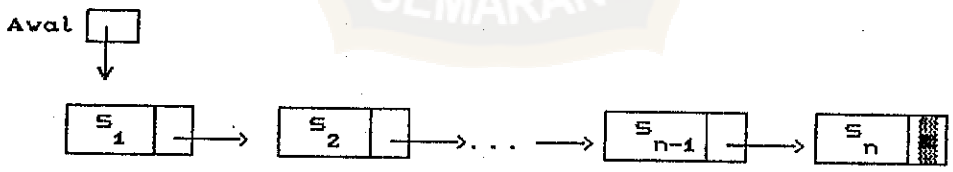
dinyatakan dengan nil. Nil adalah kata baku Pascal yang berarti bahwa pointer tidak menunjuk ke suatu simpul, dan nilainya sama dengan 0 atau bilangan negatif. Untuk mengalokasikan simpul dalam pengingat, perintah yang digunakan adalah new, yang mempunyai bentuk umum

new(peubah);

dengan peubah adalah nama peubah yang bertipe pointer. Ilustrasi mengenai linked-list ini diberikan dalam Gambar 2.1.

Setiap simpul dalam dalam suatu linked-list terdiri dari 2 field :

1. field informasi (*info field*) yang mengandung elemen simpulnya.
2. field link (*link field*) yang mengandung pointer ke pengikutnya.



Gambar 2.1. Linked-list dengan n simpul.

2.3. Operasi Linked-List

Representasi linked-list menyediakan fasilitas operasi penambahan, penghapusan dan pencarian simpul tertentu.

Untuk operasi-operasi simpul yang akan diuraikan, dimisalkan digunakan deklarasi pointer dan simpul sebagai berikut.

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

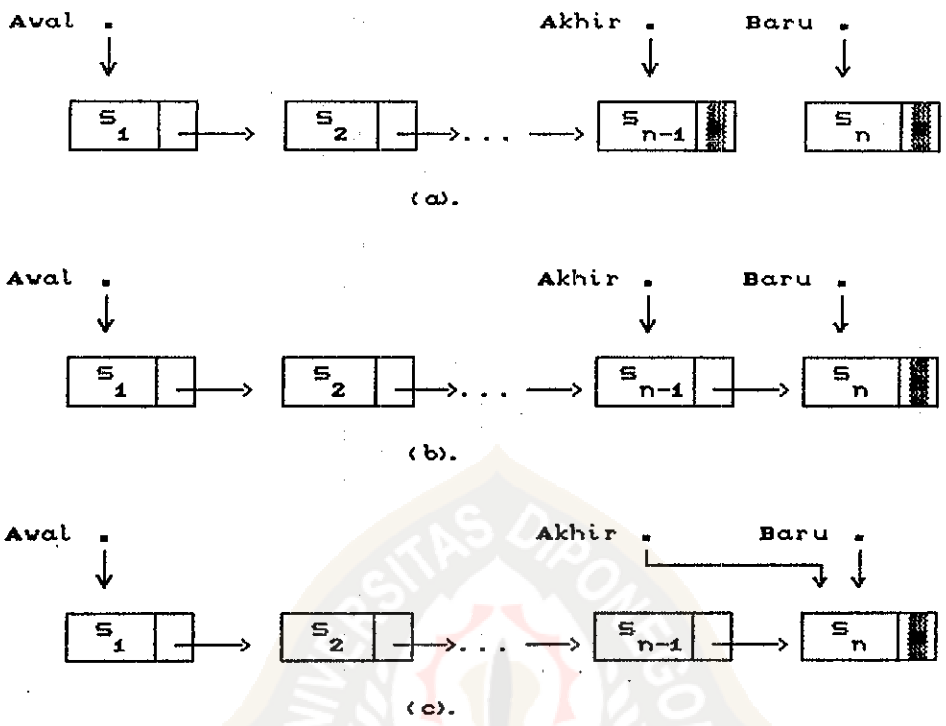
2.3.1. Menambah Simpul

Operasi menambah simpul bisa dipecah berdasarkan posisi simpul baru yang akan disisipkan, yaitu simpul baru selalu ditambahkan di belakang simpul terakhir, simpul baru selalu diletakkan sebagai simpul pertama, dan simpul baru menyisip diantara dua simpul yang sudah ada.

2.3.1.1. Menambah di Belakang

Penambahan simpul di akhir linked-list dijelaskan sebagai berikut. Simpul-simpul baru yang ditambahkan selalu akan menjadi simpul terakhir. Ilutrasi penambahannya digambarkan dalam Gambar 2.2.

Pointer Awal adalah pointer yang menunjuk ke simpul pertama, pointer Akhir menunjuk ke simpul terakhir dan simpul yang ditunjuk oleh pointer Baru adalah simpul yang akan ditambahkan (Gambar 2.2.(a)).



Gambar 2.2. Menambah simpul di belakang.

Untuk menyambung simpul yang ditunjuk oleh Akhir dan Baru, pointer pada simpul yang ditunjuk oleh simpul Akhir dibuat sama dengan Baru (Gambar 2.2.(b)), dengan perintah

```
Akhir.^Berikut := Baru;
```

Kemudian pointer Akhir dibuat sama dengan pointer Baru dengan perintah

```
Akhir := Baru;
```

(Gambar 2.2.(c)).

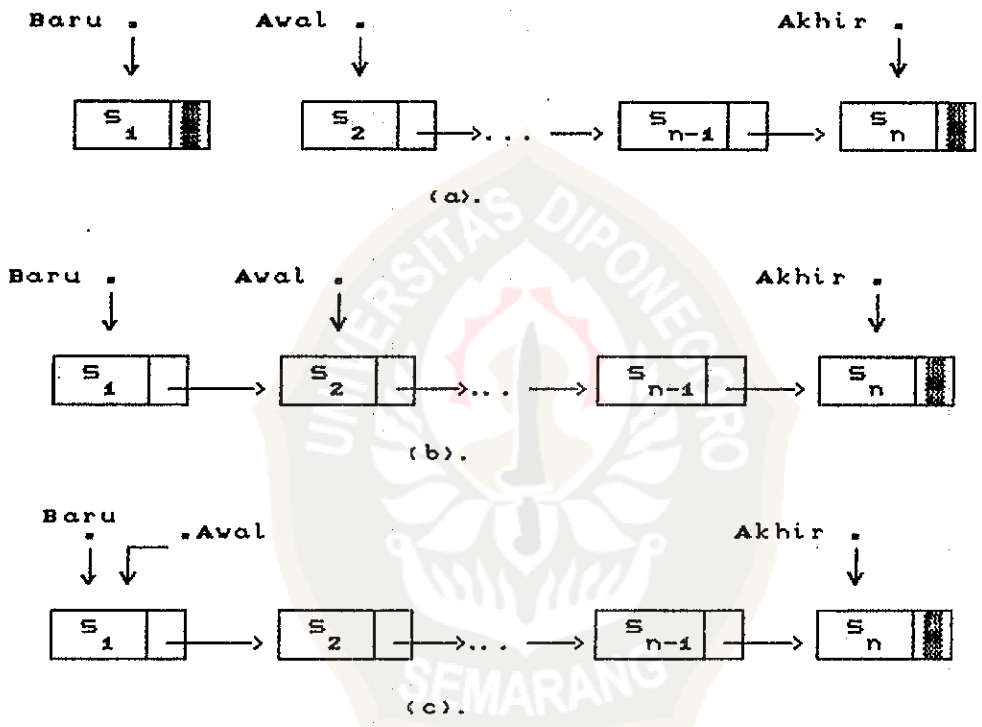
2.3.1.2. Menambah di Awal

Secara garis besar operasi penambahan simpul di awal linked-list, seperti diilustrasikan dalam Gambar 2.3. dapat dijelaskan sebagai berikut. Pertama kali pointer

pada simpul yang ditunjuk oleh pointer Baru dibuat sama dengan Awal, dengan perintah

```
Baru^.Berikut := Awal;
```

(Gambar 2.3.(b)).



Gambar 2.3. Penambahan simpul di aval Linked-list.

Kemudian Aval dibuat sama dengan Baru (Gambar 2.3.(c)), dengan pernyataan

```
Awal := Baru;
```

Dengan cara seperti itu simpul baru akan selalu diperlakukan sebagai simpil pertama dalam linked-list.

Dalam hal linked-list masih nil, perlu diadakan pengujian dengan pernyataan

```
if Awal = nil then
```

kemudian pointer Akhir dibuat sama dengan Baru.

2.3.1.3. Menambah di Tengah (Menyisipkan)

Untuk menambah simpul di tengah (menyisipkan) linked-list diperlukan bantuan sebuah pointer bantu, misalnya Bantu. Dalam hal ini simpul baru akan diletakkan setelah simpul yang ditunjuk oleh pointer Bantu dan dimisalkan bahwa linked-listnya terurut naik (*ascending ordered*). Operasi penyisipan simpul diilustrasikan dalam Gambar 2.4.

Pertama kali ditentukan dimana simpul baru akan disisipkan, yaitu dengan menempatkan pointer Bantu pada suatu tempat. Kemudian pointer pada simpul yang ditunjuk oleh Baru dibuat sama dengan pointer pada simpul yang ditunjuk oleh Bantu, dengan perintah

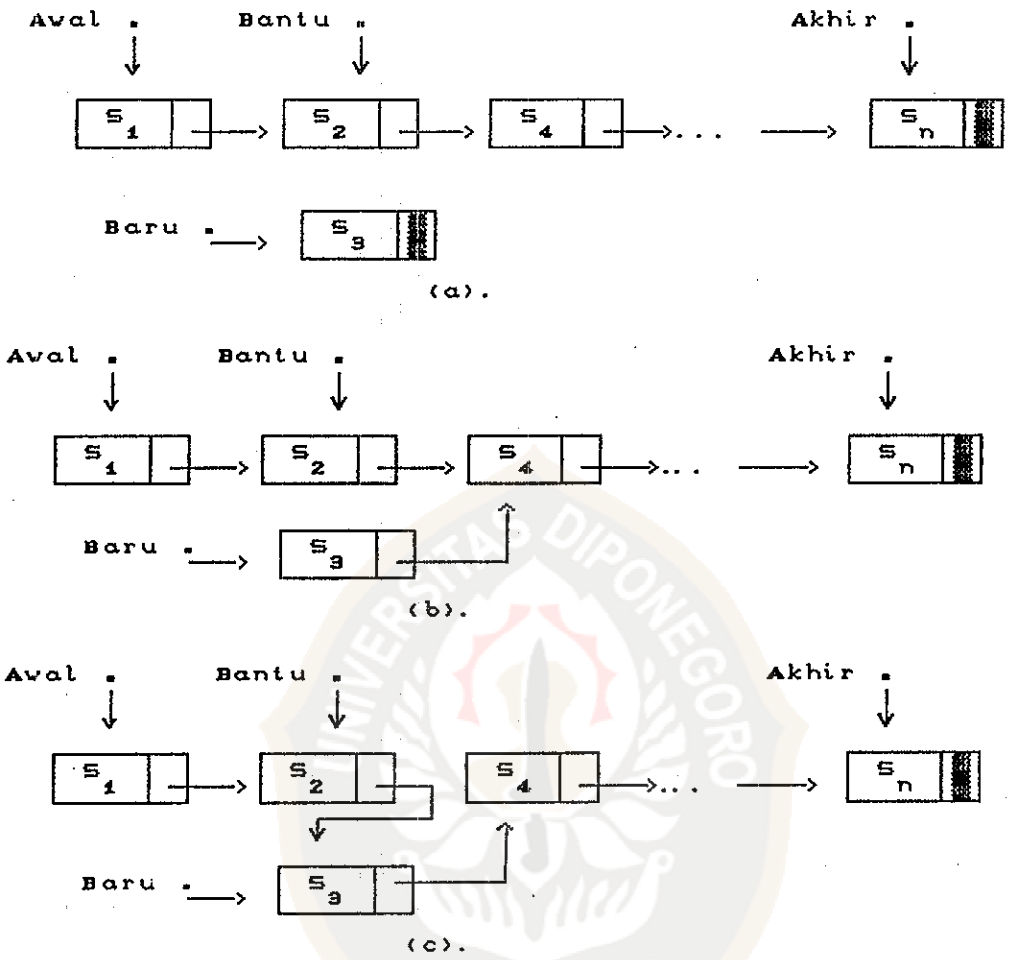
```
Baru^.Berikut := Bantu^.Berikut;
```

(Gambar 2.4.(b)). Selanjutnya pointer pada simpul yang ditunjuk oleh simpul Bantu dibuat sama dengan Baru, dengan perintah

```
Bantu^.Berikut := Baru;
```

(Gambar 2.4.(c)).

Perhatikan bahwa urutan kedua proses ini tidak boleh terbalik. Karena akan menyebabkan simpul yang ditunjuk oleh pointer pada simpul yang ditunjuk oleh pointer Bantu dan simpul-simpul sesudahnya akan terlepas dari ikatan linked-listnya. Sehingga penambahan simpul baru tidak dapat dilakukan dengan mempertahankan linked-list lama.



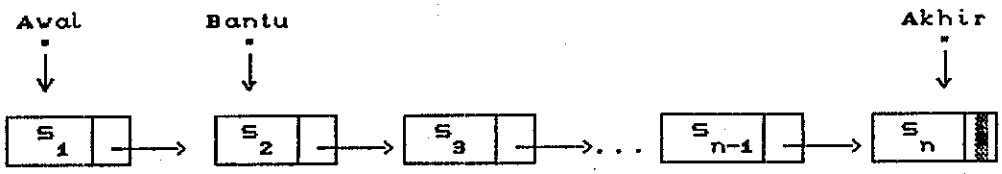
Gambar 2.4. Penyisipan simpul di tengah liked-list.

2.3.2. Menghapus Simpul

Yang perlu diperhatikan dalam menghapus simpul adalah bahwa simpul yang bisa dihapus adalah simpul yang berada sesudah simpul yang ditunjuk oleh suatu pointer, kecuali untuk simpul pertama. Dengan demikian, simpul yang ditunjuk oleh suatu pointer atau simpul sebelumnya tidak dapat dihapus.

Misalkan linked-list yang akan dihapus suatu simpulnya diilustrasikan seperti dalam Gambar 2.5. Maka simpul yang berada di sebelah kanan simpul yang ditunjuk

oleh suatu pointer, yaitu hanya simpul yang berisi S_1 dan simpul yang berisi S_3 yang dapat dihapus supaya linked-list tetap bisa dipertahankan.



Gambar 2.5. Contoh linked-list

Simpul yang berisi S_2 memang bisa dihapus, tetapi linked-listnya akan terputus dan karenanya akan lebih sulit untuk menyambung lagi simpul yang berisi S_1 dan simpul yang berisi S_3. Demikian pula apabila simpul yang berisi S_n dihapus, maka ada kemungkinan pointer pada simpul yang berisi S_{n-1} akan menjadi tidak menentu dan menyebabkan terjadinya kesalahan.

2.3.2.1. Menghapus Simpul Awal

Untuk menghapus simpul awal maka pointer Bantu dibuat sama dengan pointer Awal, dengan perintah

```
Bantu := Awal;
```

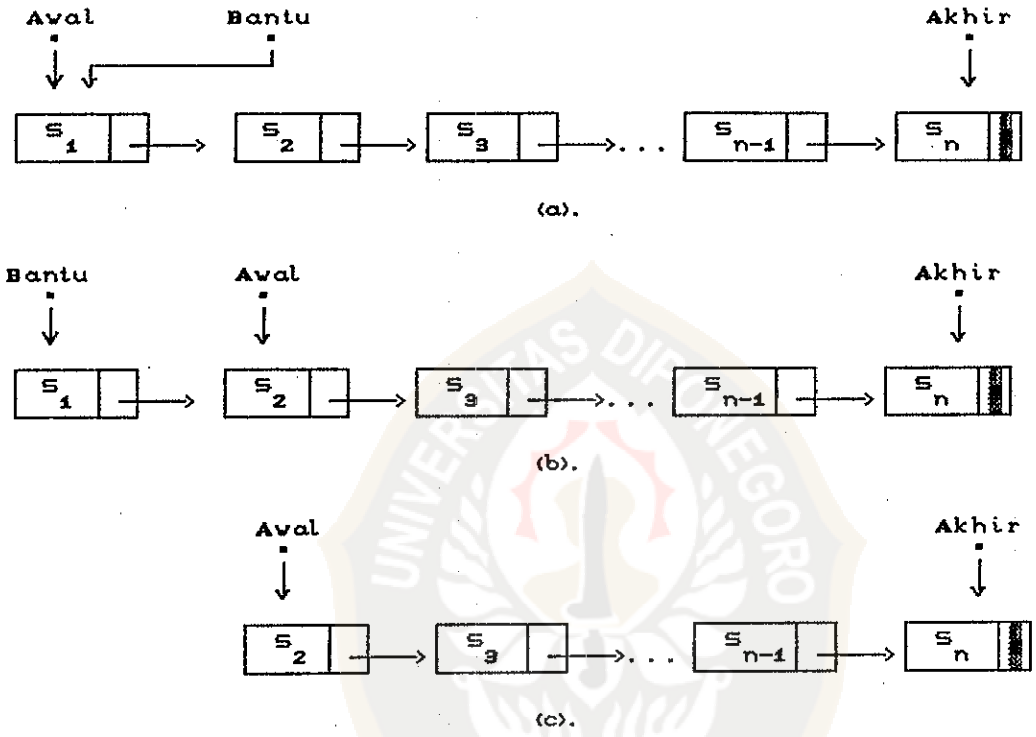
(Gambar 2.6.(a)). Kemudian pointer Awal dipindah ke simpul yang ditunjuk oleh pointer pada simpul yang ditunjuk oleh pointer Bantu, dengan perintah

```
Awal := Bantu^.Berikut;
```

(Gambar 2.6.(b)). Selanjutnya, simpul yang ditunjuk oleh pointer Bantu dihapus dengan perintah

dispose(Bantu);

(Gambar 2.6.(c)).



Gambar 2.6. Menghapus simpul aval

2.3.2.2. Menghapus Simpul di Tengah

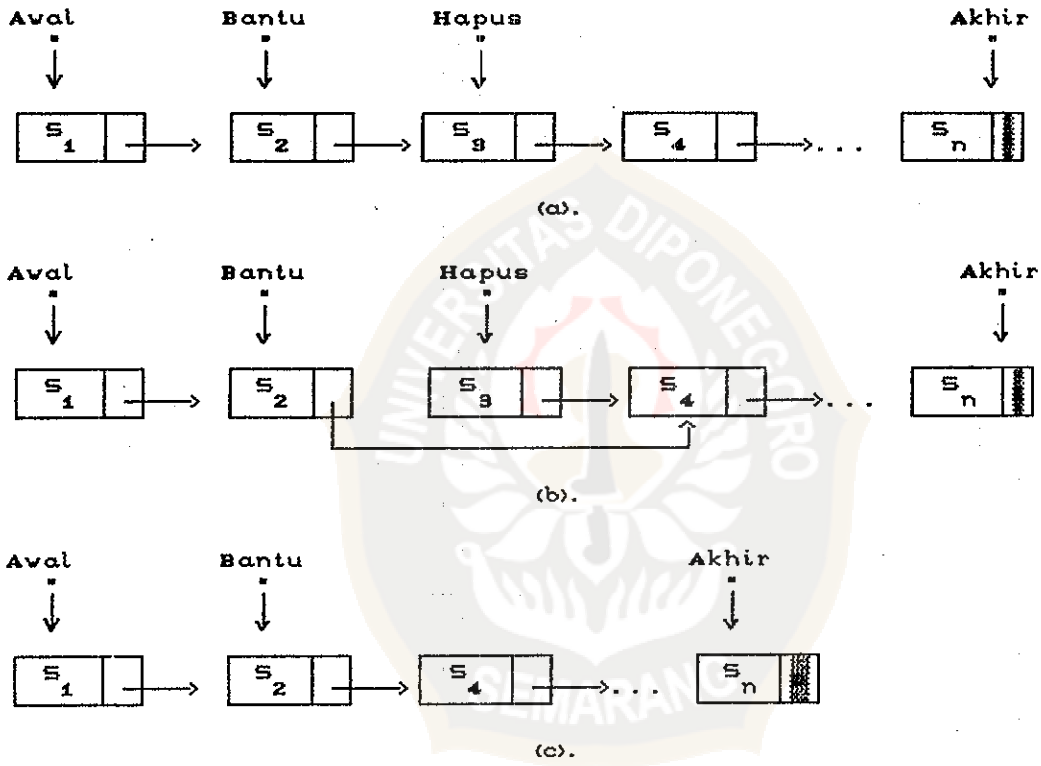
Yang pertama kali dilakukan untuk menghapus simpul di tengah suatu linked-list adalah meletakkan pointer Bantu pada simpul di sebelah kiri simpul yang akan dihapus. Simpul yang akan dihapus ditunjuk dengan pointer lain, misalnya Hapus (Gambar 2.7.(a).). Kemudian, pointer pada simpul yang ditunjuk oleh Bantu ditunjukkan pada simpul yang ditunjuk oleh pointer pada simpul yang akan dihapus, dengan perintah

Bantu^.Berikut := Hapus^.Berikut;

(Gambar 2.7.(b)). Selanjutnya simpul yang ditunjuk oleh pointer Hapus dihapus dengan perintah

dispose(Hapus);

(Gambar 2.7.(c)).



Gambar 2.7. Menghapus simpul di tengah.

2.3.2.3. Menghapus Simpul Terakhir

Proses menghapus simpul terakhir hampir sama dengan proses menghapus simpul di tengah. Pertama kali pointer Bantu diletakkan pada simpul di sebelah kiri simpul terakhir dan pointer Hapus dibuat sama dengan Akhir dengan perintah

Hapus := Akhir;

(Gambar 2.8.(a)). Kemudian pointer Akhir dibuat sama

dengan pointer Bantu dengan perintah

```
Akhir := Bantu;
```

(Gambar 2.8.(b)). Setelah itu pointer pada simpul yang ditunjuk oleh pointer Bantu dibuat nil (Gambar 2.8.(c)). Dan akhirnya simpul yang ditunjuk oleh pointer Hapus dihapus dengan perintah

```
dispose(Hapus);
```

(Gambar 2.8.(d)).

2.3.3. Mencari Simpul

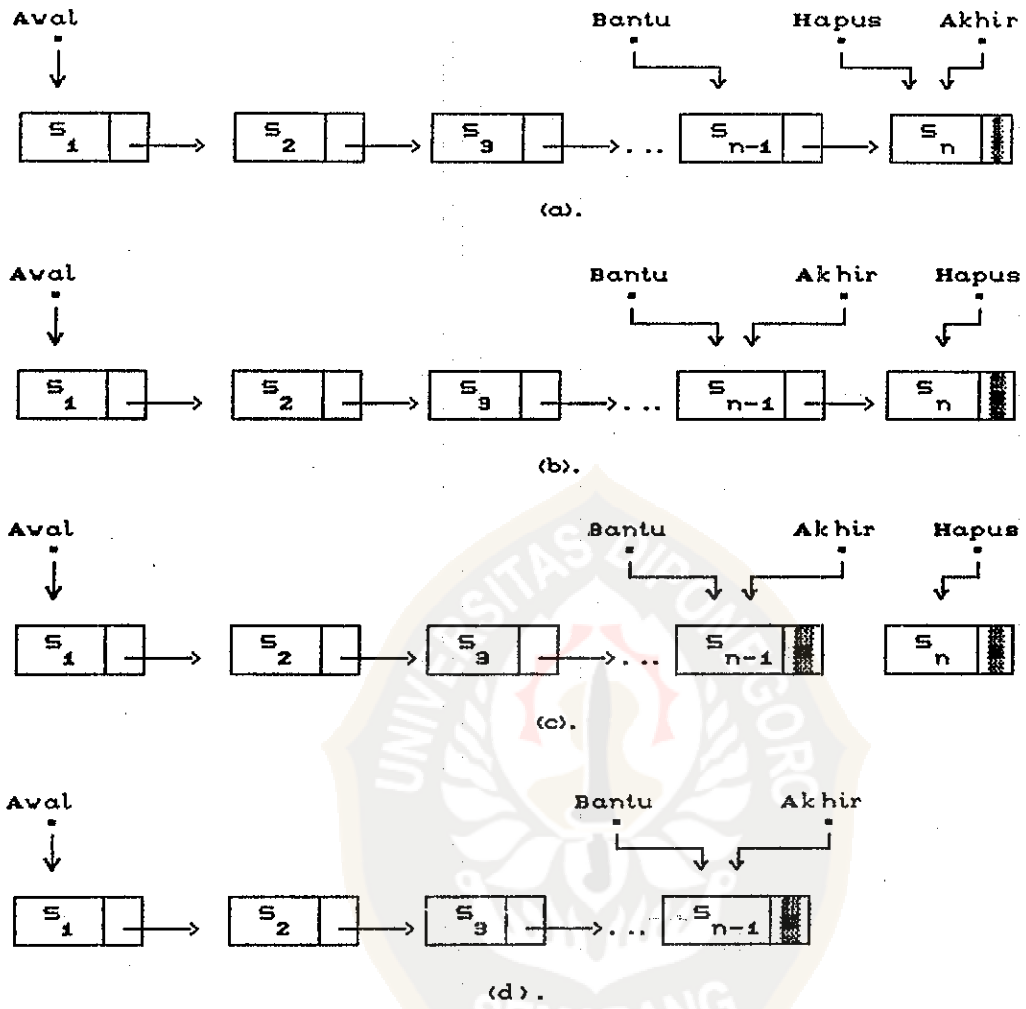
Pencarian simpul dari suatu linked-list pada dasarnya adalah dimaksudkan untuk mencari data / elemen dalam field informasi simpul itu. Oleh karena itu, proses pencarian data yang diuraikan dalam bagian ini berarti juga pencarian simpul yang mengandung data itu.

Misalkan linked-list dalam pencarian data diilustrasikan seperti pada Gambar 2.5., dan misalkan data yang dicari disimpan dalan peubah Elemen. Maka proses pencarian data dapat dijelaskan sebagai berikut. Pertama kali, pointer Bantu dibuat sama dengan pointer Awal, dengan perintah

```
Bantu := Awal;
```

Kemudian isi simpul yang ditunjuk oleh pointer Bantu dibandingkan dengan Elemen, dengan pernyataan

```
if Bantu^.Info = Elemen then
```



Gambar 2.8. Menghapus simpul terakhir.

Apabila belum sama, pointer Bantu dipindah ke simpul sebelah kanannya dengan perintah

```
Bantu := Bantu^.Berikut;
```

dan proses pembandingan diulangi lagi.

Proses ini dilanjutkan sampai dapat ditentukan apakah Elemen ada dalam linked-list.

Dalam hal linked-listnya tidak dalam keadaan terurut (*unordered linked-list*), keadaan yang paling buruk (*worst case*) terjadi apabila pencarian harus terus dilakukan

sampai akhir linked-list tercapai.

Apabila linked-list yang akan dicari datanya sudah dalam keadaan terurut, misalnya terurut turun (*descending order*) maka jika nilai Elemen telah lebih besar dari nilai data dalam setiap field informasi simpulnya, proses pencarian bisa dihentikan tanpa harus menelusuri ke seluruh linked-list.

2.4. Linked-List Berkepala

Untuk tujuan-tujuan tertentu biasanya perlu meletakkan sebuah simpul awal yang tidak berisi informasi seperti halnya simpul-simpul lain dalam linked-list, tetapi keberadaannya sangat diperlukan untuk lebih mempercepat proses eksekusi. Simpul yang demikian disebut simpul kepala (*headed node*) sehingga linked-listnya disebut linked-list berkepala (*headed linked-list*).

Dengan adanya simpul kepala, maka dibutuhkan satu prosedur sederhana untuk proses inisialisasi linked-list untuk membentuk simpul kepala.

2.5. Algoritma

Algoritma didefinisikan sebagai himpunan berhingga instruksi-instruksi yang apabila ditelusuri, menyelesaikan suatu tugas khusus. Lagi pula, semua algoritma harus memenuhi kriteria berikut :

1. Adanya masukan (*Input*)

   Suatu algoritma harus mempunyai kuantitas nol atau

lebih yang disuplai secara eksternal.

2. Adanya keluaran (*Output*)

   Input suatu algoritma harus memberikan hasil sekurang-kurangnya satu kuantitas.

3. Kepastian (*Definiteness*)

   Tiap-tiap instruksi dalam algoritma harus jelas dan tidak mempunyai arti ganda (*Unumbiguous*).

4. Keberhinggaan (*Finiteness*)

   Apabila instruksi-instruksi suatu algoritma ditelusuri, maka untuk semua kasus, algoritma berhenti setelah sejumlah berhingga langkah-langkah.

5. Keefektifan (*Effektiveness*)

   Setiap instruksi harus cukup mendasar untuk dilaksanakan (langsung mengarah kesasaran yang dituju). Tidaklah cukup bahwa setiap operasi adalah definite seperti dalam (3), tetapi juga harus fisibel (*feasible*) yakni dapat dikerjakan dengan mudah.

Tentu saja, algoritma yang terpakai disini dipikirkan dari sudut pandang Ilmu Komputer.