ALGORITMA PENYISIPAN

Terdiri dari :
1. Algoritma Penyisipan Simpul pada bagian awal list.
2. Algoritma Penyisipan Simpul sesudah suatu simpul yang diketahui lokasinya.
3. Algoritma Penyisipan Simpul ke dalam suatu list terurut.

Semua algoritma diasumsikan bahwa Linked List tersimpan di dalam memori dalam bentuk

LIST(INFO, LINK, START, AVAIL)

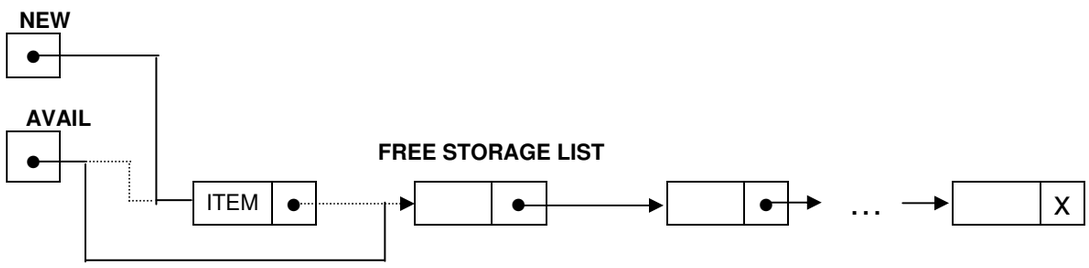
dari variabel ITEM menyatakan informasi baru yang akan ditambahkan ke dalam list. Karena semua Algoritma Penyisipan kita tersebut membutuhkan simpul dari list AVAIL, maka selalu mengandung langkah :
1. Memeriksa ruang bebas dari list AVAIL, kalau ternyata tidak ada lagi, yakni dalam hal ini AVAIL = NULL, Algoritma mengirim pesan OVERFLOW.
2. Pemindahan simpul pertama dari list AVAIL, menggunakan variabel NEW. Pemindahan ini melalui sepasang statement

   NEW := AVAIL
   AVAIL := LINK(AVAIL)

3. Menyalin informasi baru tersebut ke dalam simpul baru, atau dengan perkataan lain, memakai statement

   INFO(NEW) := ITEM

Diagram skematik untuk langkah 2 dan 3 adalah (Gambar 15a):



Gambar 15a

PENYISIPAN PADA BAGIAN AWAL LIST

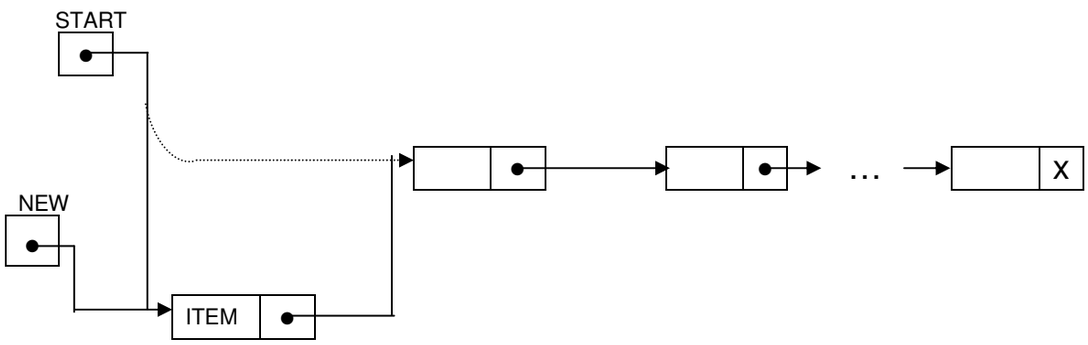
Linked list tidak perlu terurut dan penyisipan tidak harus di suatu posisi tertentu. Maka posisi termudah untuk memasukkan simpul baru adalah di bagian awal list.

ALGORITMA

INSFIRST(INFO, LINK, START, AVAIL, ITEM)
Algoritma ini menyisipkan ITEM sebagai simpul pertama dari list

1. [Apakah Overflow?] Jika AVAIL = NULL, maka tulis OVERFLOW, dan Exit.
2. [Memindahkan simpul pertama dari list AVAIL.]
   NEW := AVAIL dan AVAIL := LINK[AVAIL].
3. INFO[NEW] := ITEM. [Menyalin data baru ke dalam simpul baru.]
4. LINK[NEW] := START. [Simpul baru sekarang menuding ke simpul semula.]
5. START := NEW. [Mengubah START agar menuding ke simpul yang baru.]
6. Exit.

Diagram skematik langkah 2 dan 3 terlihat pada Gambar 15a.
Diagram skematik langkah 4 dan 5 dapat dilihat pada Gambar 15b.



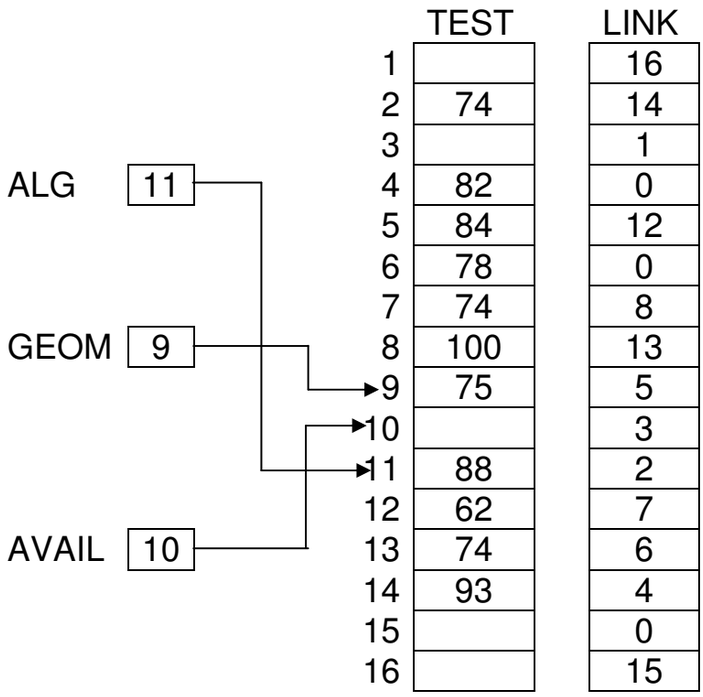
Gambar 15b

Contoh :
Pandang list pada Gambar 7 yang lalu.
Misalkan angka 75 akan disisipkan pada awal dari List Geometri.
Dengan menggunakan algoritma di atas, perhatikan bahwa ITEM =75, INFO = TEST, dan START = GEOM.

INSFIRST(INFO, LINK, START, AVAIL, ITEM)

1. Karena AVAIL <> NULL, ke langkah 2.
2. NEW = 9, maka AVAIL = LINK[9] = 10.
3. TEST[9] = 75.
4. LINK[9] = 5.
5. GEOM = 9.
6. Exit.

Gambar 16 memperlihatkan keadaan setelah 75 dimasukkan ke dalam List Geomaetri.
Perhatikan bahwa hanya 3 penuding yang berubah, yaitu AVAIL, GEOM dan LINK[9].



Gambar 16

PENYISIPAN SESUDAH SIMPUL DIKETAHUI

Pandang bahwa nilai dari LOC, yaitu lokasi dari simpul A di dalam linked list, atau nilai LOC = NULL. Berikut ini adalah Algoritma untuk menyisipkan ITEM ke dalam list, tepat sesudah simpul A, atau jika LOC = NULL, maka ITEM disisipkan sebagai simpul pertama dari list.

Misalkan N adalah simpul baru, yang lokasinya adalah NEW. Jika LOC = NULL, maka penyisipan dikerjakan seperti pada algoritma yang lalu. Dalam hal lain, seperti terlihat pada gambar 9, jadikan N menuding simpul B (simpul yang sebelumnya mengikuti simpul A), dengan menggunakan statement

LINK[NEW] := LINK[LOC]

Dan jadikan simpul A menuding simpul baru N dengan menggunakan statement

LINK[LOC] := NEW

Algoritma : INSLOC(INFO,LINK,START,AVAIL<LOC<ITEM)
{Algoritma ini dimaksudkan untuk menyisipkan ITEM, sehingga mengikuti simpul dengan lokasiLOC atau menyisipkan ITEM sebagai Simpul pertama, bila LOC = NULL.

1. [OVERFLOW ?] jika AVAIL = NULL, maka : Tulis: OVERFLOW, dan EXIT
2. [Memindahkan Simpul pertama dari List AVAIL]
   NEW := AVAIL
   AVAIL := LINK[AVAIL]
3. INFO[NEW] := ITEM. [Menyalin data baru ke dalam simpul baru]
4. Jika LOC = NULL, maka : [Sisipkan sebagai Simpul pertama]
   LINK[NEW] := START dan START := NEW.
   Dalam hal lain : [Sisipkan sesudah Simpul dengan lokasi LOC]
   LINK[NEW] := LINK[LOC] dan LINK[LOC] := NEW.
   [Akhir dari struktur jika]
5. EXIT

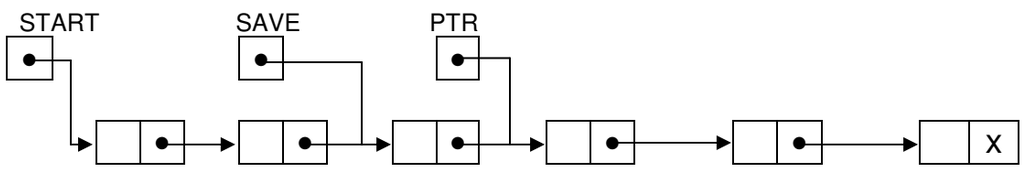
PENYISIPAN KE DALAM LINKED LIST TERURUT

Misalkan ITEM akan disisipkan ke dalam Linked List terurut LIST. ITEM harus disisipkan di antara simpul A dan B yang berturutan sedemikian sehingga

INFO(A) < ITEM < INFO(B)

Prosedur untuk mencari lokasi simpul A, yakni mencari lokasi dari simpul terakhir dari List yang nilainya kurang dari ITEM. Lakukan traversal list, pergunakan variabel penuding PTR, dan bandingkan ITEM dengan INFO(PTR) pada masing-masing simpul. Selama traversal, jaga jejak lokasi simpul sebelumnya dengan menggunakan variabel penuding SAVE (Gambar 17). Maka SAVE dan PTR ter-update dengan statement

SAVE := PTR dan PTR := LINK[PTR]



Gambar 17

Traversal dilanjutkan selama INFO[PTR] < ITEM, atau akan berhenti saat ITEM <= INFO[PTR]. Kemudian PTR menuding ke simpul B, maka SAVE berisi lokasi dari simpul A.

Kasus bahwa list adalah hampa, dan ITEM < INFO[START], sehingga LOC = NULL ditangani sendiri, karena tidak mengandung variabel SAVE.
Prosedur : FINDA (INFO, LINK, START, ITEM, LOC)
{Prosedur ini akan menemukan lokasi LOC dari simpul terakhir dalam list terurut, sedemikian sehingga INFO[LOC] < ITEM, atau menjadikan LOC = NULL}

1. [List hampa?] Jika START = NULL, maka LOC := NULL, dan Return.
2. [Kasus khusus?] Jika ITEM < INFO[START], maka LOC := NULL, dan Return.
3. SAVE := START dan PTR := LINK[START]. [Mengawali Penuding]
4. Ulangi langkah 5 dan 6 sementara PTR <> NULL.
5. Jika ITEM < INFO[PTR], maka :
   LOC := SAVE, dan Return.
   [Akhir dari struktur jika]
6. SAVE := PTR dan PTR := LINK[PTR]. [Mengupdate Penuding].
   [Akhir dari loop langkah 4]
7. LOC := SAVE
8. Return.

Sekarang kita mempunyai Algoritma untuk menyisipkan ITEM ke dalam linked list terurut List, dengan memanfaatkan 2 prosedur terakhir.

Algoritma : INSSRT(INFO, LINK, START, AVAIL, ITEM)
{Algoritma ini menyisipkan ITEM ke dalam suatu linked list terurut}

1. [Gunakan Prosedur FINDA untuk mencari lokasi dari simpul sebelum ITEM]
   Call FINDA(INFO, LINK, START, ITEM, LOC).
2. [Gunakan Algoritma INSLOC untuk menyisipkan ITEM sesudah simpul dengan lokasi LOC]
   Call INSLOC(INFO, LINK, START, AVAIL, LOC, ITEM).
3. Exit.

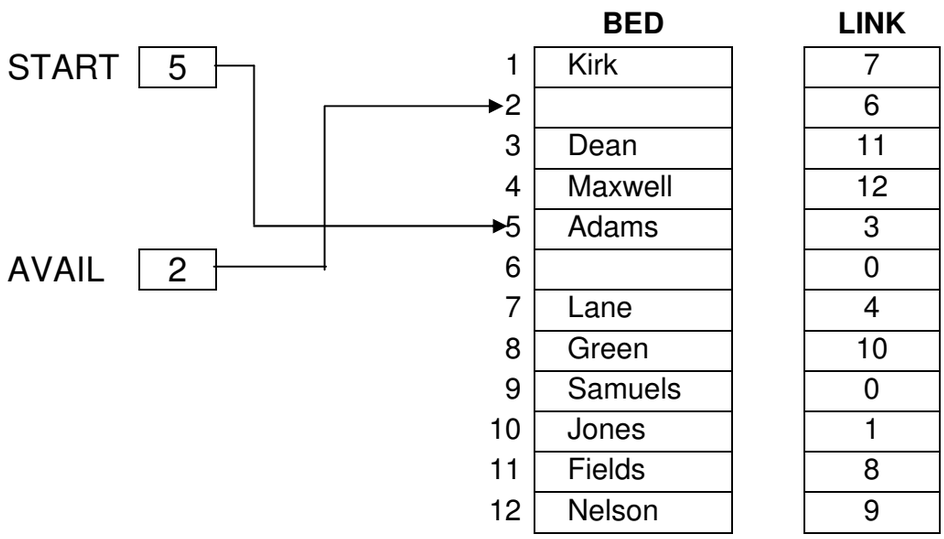
Contoh :
Perhatikan list yang terurut secara alfabetik dari para Pasien pada Gambar 6 yang lalu. Pandang bahwa Jones akan disisipkan ke dalam List tersebut. Gunakan Algoritma INSSRT di atas, yang pada hakikatnya melaksanakan prosedur FINDA, dan kemudian Algoritma INSLOC. Perhatikan bahwa ITEM = Jones, dan INFO = BED.

a. FINDA(BED, LINK, START, ITEM, LOC)
   1. Karena START <> NULL, maka kita melangkah ke langkah 2.
   2. Karena BED[5] = Adams < Jones, kendali beralih ke langkah 3.
   3. SAVE = 5, dan PTR = LINK[5] = 3

4. Langkah 5 dan 6 diulang, sebagai berikut :
   a. BED[3] = Dean < Jones, maka SAVE = 3, dan PTR = LINK[3] = 11
   b. BED[11] = Fields < Jones, maka SAVE = 11, dan PTR = LINK[11] = 8
   c. BED[8] = Green < Jones, maka SAVE = 8, dan PTR = LINK[8] = 1
   d. Karena BED[1] = Kirk > Jones, maka LOC = SAVE = 8, dan Return.

b. INSLOC(BED, LINK, START, AVAIL, LOC, ITEM)
   1. Karena AVAIL <> NULL, maka kita melangkah ke langkah 2.
   2. NEW = 10, dan AVAIL = LINK[10] = 2
   3. BED[10] = Jones
   4. Karena LOC <> NULL, maka
      LINK[10] = LINK[8] = 1, dan LINK[8] = NEW = 10
   5. Exit

Gambar 18 menunjukkan struktur data sesudah Jones disisipkan ke dalam List. Di sini hanya 3 penuding yang berubah, yakni AVAIL, LINK[10] dan LINK[8].



Gambar 18

PENGHAPUSAN SIMPUL LINKED LIST

Misalkan simpul N adalah simpul dari List yang terletak di antara simpul A dan simpul B (Gambar 19a). Simpul N tersebut akan dihapus dari List. Diagram skematik dari penghapusan terlihat pada Gambar 19b.

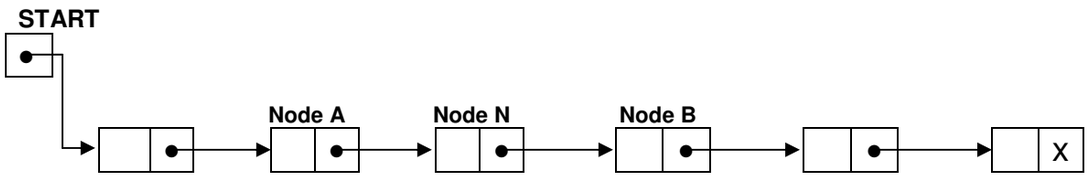
Penghapusan terjadi begitu nextpointer dari A berubah menuding ke B. Dalam penghapusan ini, kita harus mengingat alamat dari simpul A, simpul pendahulu dari simpul yang akan dihapus tersebut.

Pandang linked list tersimpan di memori dalam bentuk :

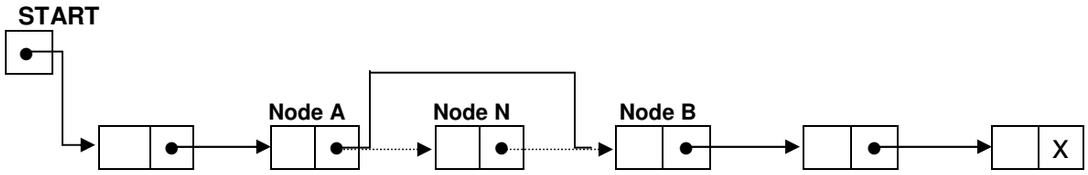
LIST(INFO, LINK, START, AVAIL)

Gambar 19 tidak memperlihatkan fakta bahwa kita melakukan penghapusan simpul N, dan akan memulangkan ruang memori kepada list AVAIL.
Diagram skematik yang lebih tepat terlihat pada Gambar 20. Perhatikan bahwa 3 field penuding berubah, yakni :

1. Nextpointer dari simpul A sekarang menuding ke B, yang tadinya dituding oleh N.
2. Nextpointer dari simpul N sekarang menuding ke simpul pertama dari ruang bebas, yang tadinya dituding oleh AVAIL.
3. AVAIL sekarang menuding ke simpul N.



(a) Sebelum Penghapusan

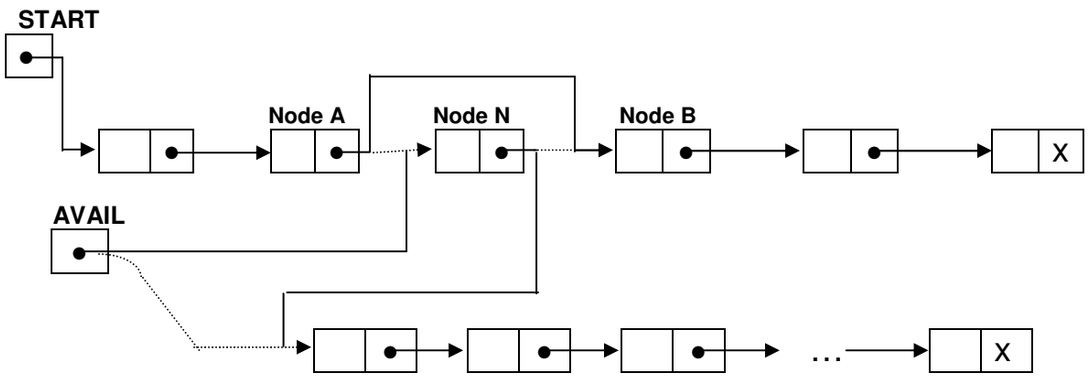


(b) Sesudah Penghapusan

Gambar 19

Di sini terdapat 2 kasus istimewa yaitu :

1. Penghapusan simpul N sebagai simpul pertama dalam list. Dalam hal ini, START akan menuding ke simpul B.
2. Penghapusan simpul N sebagai simpul terakhir dari list, simpul A akan berisi penuding NULL.

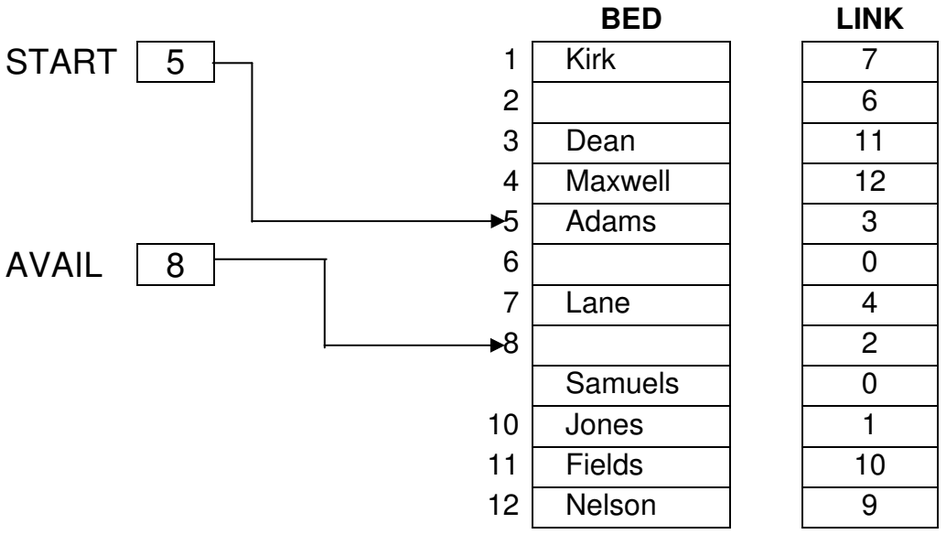


Gambar 20

Contoh :
Perhatikan list Pasien pada Gambar 18. Misalkan Pasien Green sudah diperbolehkan pulang, maka BED[8] sekarang menjadi kosong. Agar linked list tersebut tetap terjaga, maka perlu dilakukan perubahan terhadap beberapa field penuding, yakni sebagai berikut :

LINK[11] = 10 LINK[8] = 2 AVAIL = 8

Dari perubahan pertama, Fields yang tadinya mendahului Green, sekarang menuding ke Jones, yang tadinya mengikuti Green. Preubahan kedua dan ketiga akan menambah jumlah ranjang kosong baru pada list AVAIL. Ditekankan bahwa sebelum membuat penghapusan, harus mencari simpul BED[11], yang tadinya menuding ke simpul yang dihapus, BED[8].



Gambar 21

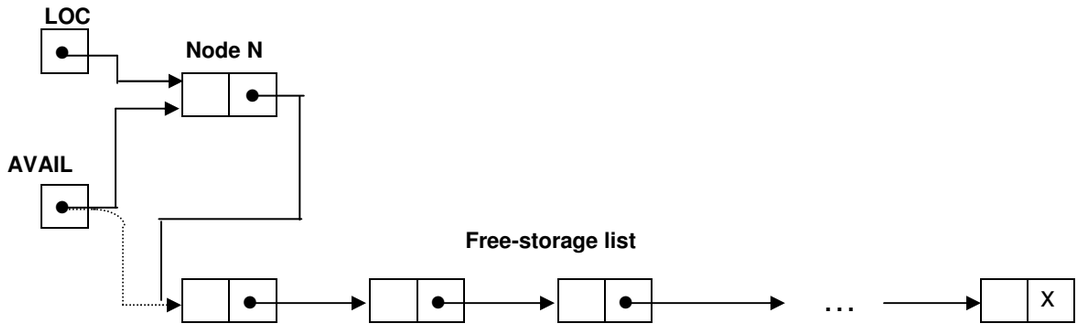
ALGORITMA PENGHAPUSAN

1. Penghapusan simpul sesudah suatu simpul yang diketahui.
2. Penghapusan simpul yang diketahui mempunyai informasi ITEM.

Asumsi : Linked list dalam bentuk LIST(INFO, LINK, START, AVAIL). Semua algoritma penghapusan selalu mengembalikan ruang memori dari simpul yang dihapus, ke bagian awal dari list AVAIL.
Karenanya, Algoritma mengandung pasangan statement berikut ini:

```
LINK[LOC] := AVAIL
AVAIL := LOC
```

Di sini LOC adalah lokasi dari simpul N yang dihapus.
Kedua operasi di atas digambarkan pada Gambar 22.



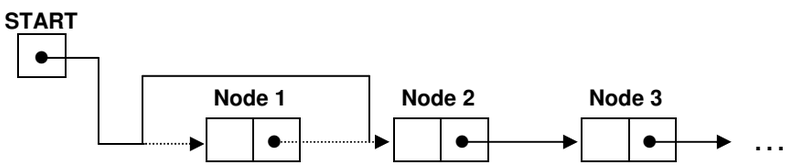
Gambar 22

PENGHAPUSAN SIMPUL SESUDAH SIMPUL YANG DIKETAHUI

Algoritma : DEL(INFO, LINK, START, AVAIL, LOC, LOCP)
{Algoritma ini dimaksudkan untuk menghapus simpul N dengan lokasi LOC. LOCP adalah lokasi dari simpul yang mendahului N atau, apabila N adalah simpul pertama, LOCP = NULL}.

1. Jika LOCP = NULL, maka :
   START := LINK[START]. [Menghapus simpul pertama]
   Dalam hal lain :
   LINK[LOCP] := LINK[LOC]. [Menghapus simpul N]
   [Akhir dari struktur jika]
2. [Mengembalikan simpul terhapus kepada list AVAIL]
3. Exit.

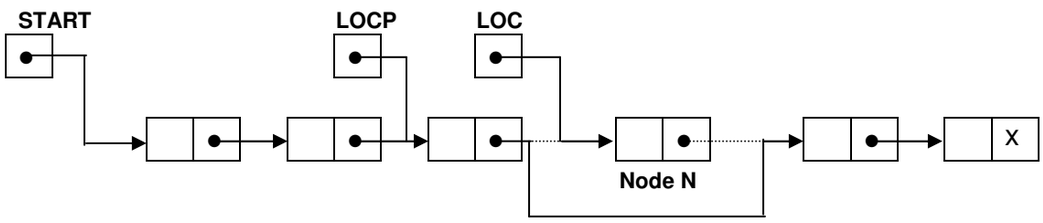
Gambar 23 merupakan diagram skematik dari statement
START := LINK[START]



Gambar 23

Gambar 24 merupakan diagram skematik dari statement

LINK[LOCP] := LINK[LOC]



Gambar 24

PENGHAPUSAN SIMPUL SESUDAH SIMPUL YANG DIKETAHUI INFORMASINYA

Algoritmanya yakni Algoritma DELLOC bekerja dengan memanfaatkan Prosedur FINDB.

Prosedur : FINDB(INFO, LINK, START, ITEM, LOC, LOCP)
{Prosedur ini dimaksudkan untuk mencari lokasi LOC dari simpul N pertama yang mengandung ITEM dan lokasi LOCP dari simpul pendahulu N. Jika ITEM tidak terdapat pada List, maka prosedur akan menjadikan LOC = NULL, dan jika ITEM muncul pada simpul pertama, maka LOCP = NULL}.

1. [List hampa?] Jika START = NULL, maka :
   LOC := NULL dan LOCP := NULL, dan Return.
   [Akhir dari struktur jika]
2. [ITEM di dalam simpul pertama?] Jika INFO[START] = ITEM, maka :
   LOC := START dan LOCP := NULL, dan Return.
3. SAVE := START dan PTR := LINK[START] [Inisialisasi Penuding]
4. Ulangi langkah 5 dan 6 sementara PTR <> NULL.
5. Jika INFO[PTR] = ITEM, maka :
   LOC := PTR dan LOCP := SAVE, dan Return.
   [Akhir dari struktur jika]
6. SAVE := PTR dan PTR := LINK[PTR] [Mengupdate Penuding]
   [Akhir dari Loop langkah 4]
7. LOC := NULL [Cari tidak berhasil]
8. Return.

Algoritma : DELLOC(INFO, LINK, START, AVAIL, ITEM)

{Algoritma ini dimaksudkan untuk menghapus dari suatu linked list, simpul N pertama yang berisi informasi ITEM}

1. [Gunakan Prosedur FINDB, untuk mencari lokasi dari N dan simpul pendahulunya]
   Call FINDB(INFO, LINK, START, ITEM, LOC, LOCP)
2. Jika LOC = NULL, maka : Tulis : ITEM tidak dalam list, dan Exit.
3. [Hapus simpul]
   Jika LOCP = NULL, maka :
      START := LINK[START]. [Hapus simpul pertama]
   Jika tidak :
      LINK[LOCP] := LINK[LOC].
   [Akhir dari struktur jika]
4. [Mengembalikan simpul terhapus ke list AVAIL]
   LINK[LOC] := AVAIL dan AVAIL := LOC.
5. Exit.

HEADER LINKED LIST

Header linked list adalah suatu list yang mengandung suatu simpul khusus yang terletak pada bagian awal dari list yang disebut simpul header. Berikut ini 2 jenis header linked list yang biasa digunakan yakni :

1. **Grounded Header List :** header list yang simpul terakhirnya berisi penuding NULL

2. **Circular Header List :** header list yang simpul terakhirnya menuding ke simpul header dari list tersebut.

LINK[START] = NULL ➔ menunjukkan Grounded Header List hampa
LINK[START] = START ➔ menunjukkan Circular Header List hampa