STRUKTUR DATA & ALGORITMA

SESI 6 – *LINKED LIST*

CATUR NUGROHO, S.KOM., M.KOM

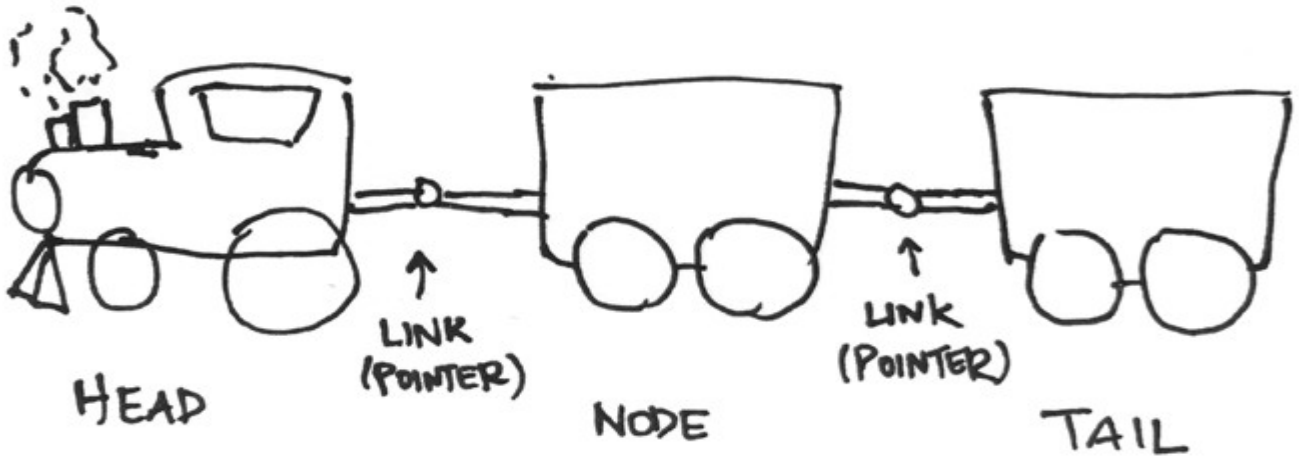
Linked List

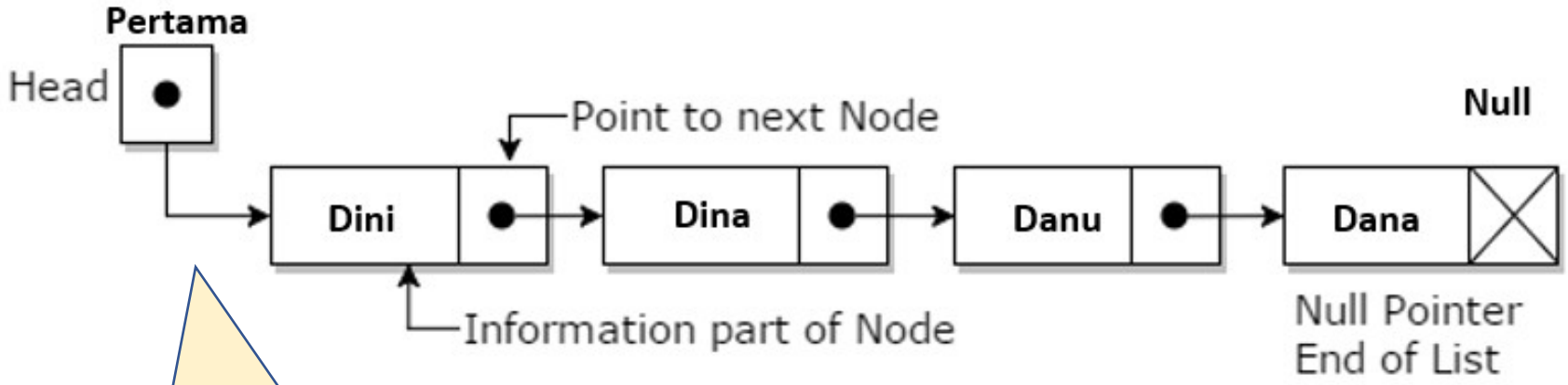
Senarai List (*Linked List*) adalah jenis struktur data yang berisi data yang disusun secaralinear dengan masing – masing disimpan dalam sebuah simpul dan antara satu simpul dengan simpul lain yang di hubungkan melalui pointer

Linked List: sejumlah obyek/ data dalam suatu kelompok yang saling berhubungan (linked) sehingga membentuk suatu urutan/ rangkaian (list) tertentu.

Struktur data ini mempuyai bentuk dasar dengan sifat data disisipkan kedalam linked list melalui salah satu ujung.

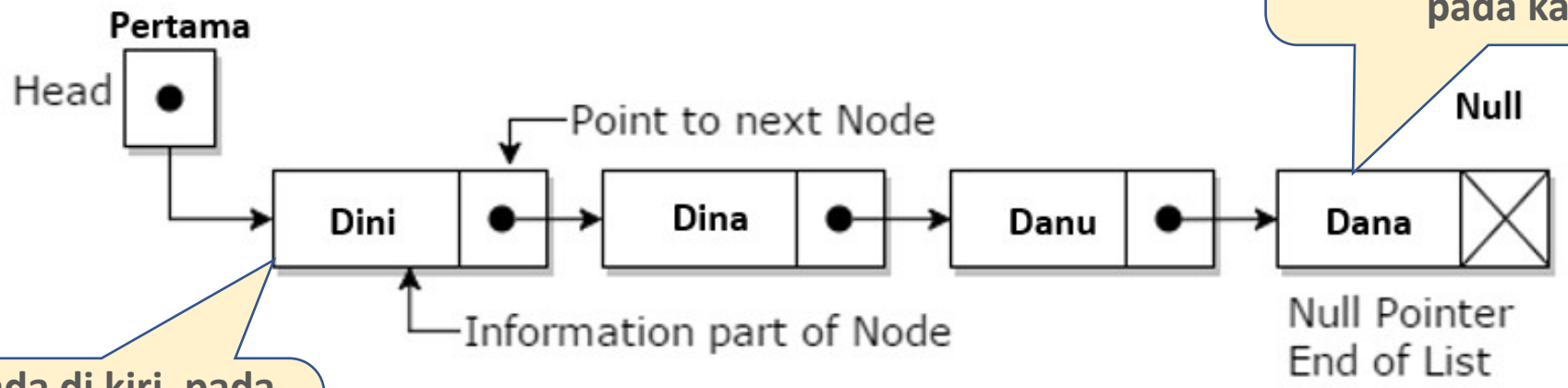
Linked List





Lihat penjelasanya

Linked List



Dana merupakan data yang pertama dimasukan pada kanan

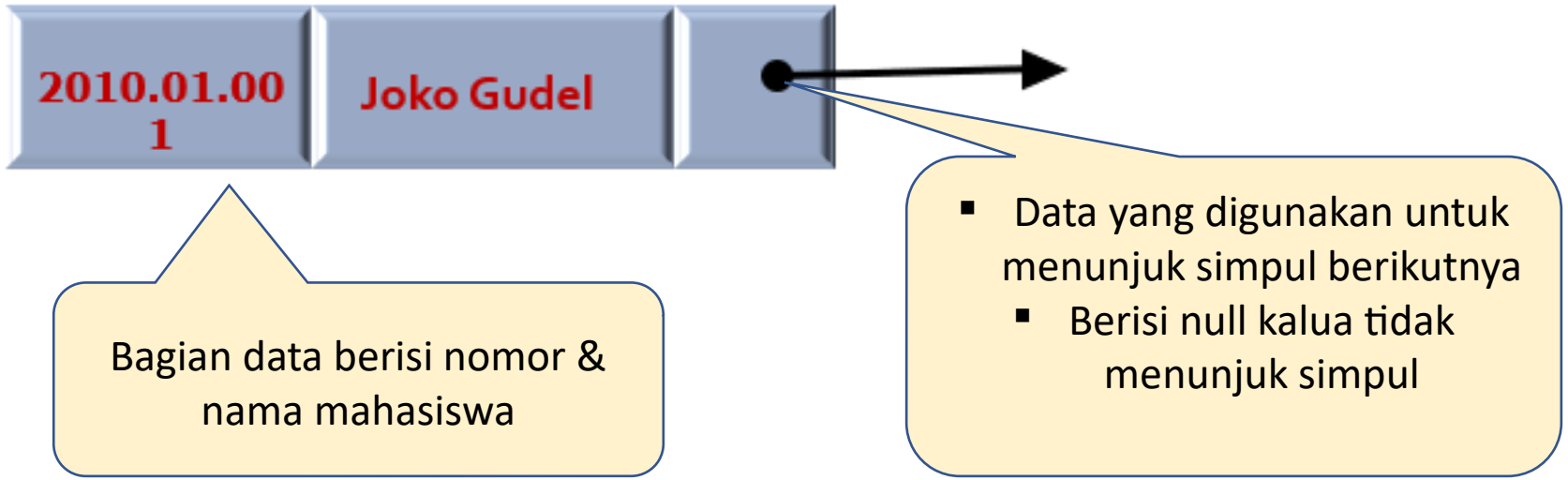
Dini berada di kiri, pada variable pertama / menunjuk data yang terakhir di input

Cara ini mirip dengan stack, yang menaruh data baru di bagian atas, bedanya linked list untuk menghapus data dapat di lakukan dimana saja

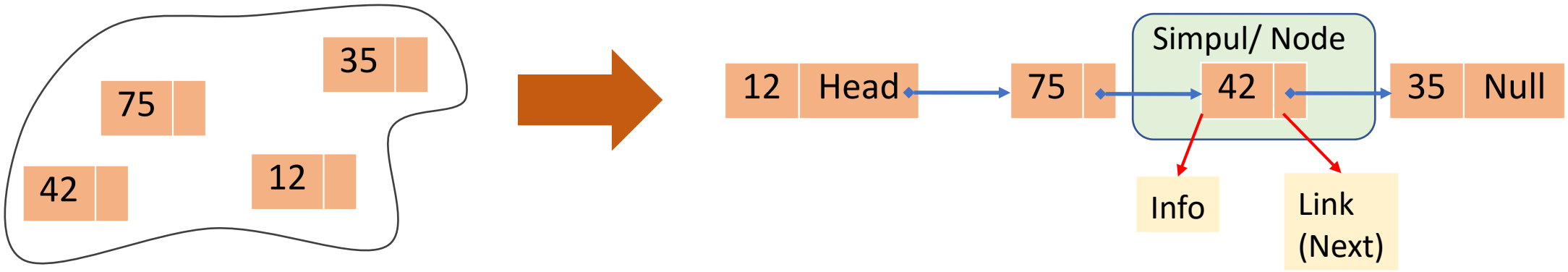
Gambar Linked List diatas memiliki 4 simpul, setiap simpul
Terdiri 2 bagian, bagian data dan bagian penujuk simpul berikutnya
Bagian data mengandung sebuah data, yaitu nama orang

Linked List

Pada *Linked List* setiap data diletakan dalam sebuah simpul (***node***)



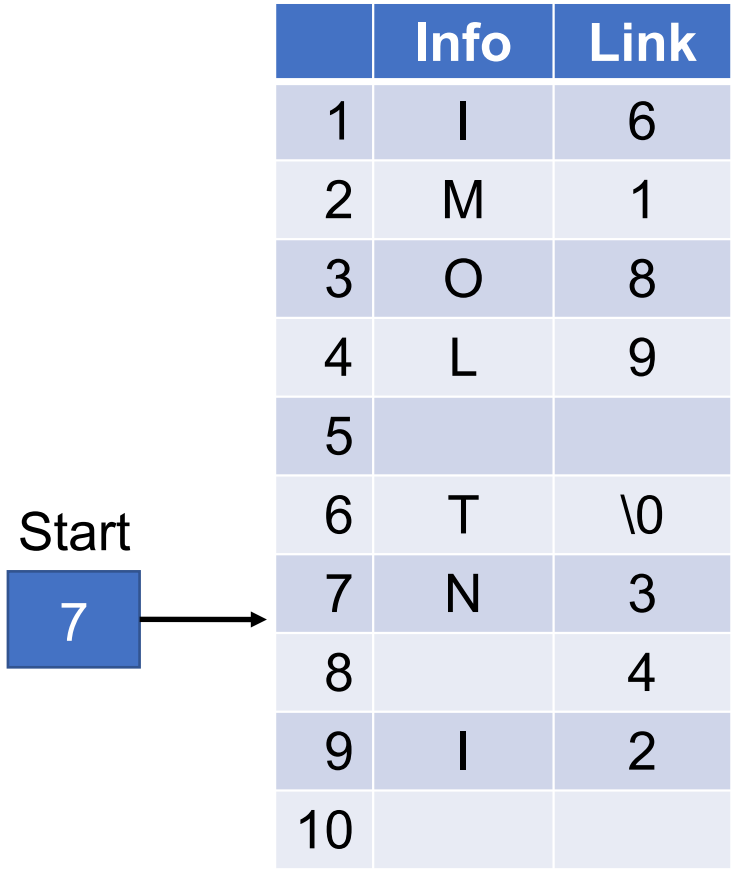
Linked List



- Setiap objek disebut Simpul (***Vertex/ Node***) yang terdiri dari 2 elemen, yaitu:
  - Elemen pertama disebut Info. Info berisi data (contoh: 12,75,42, 35).
  - Elemen kedua disebut Link/ Next yang bertipe pointer dan menunjuk ke Simpul berikutnya. (dapat juga diilustrasikan berupa anak panah).
- Elemen awal diakses oleh ***Head.***
- Link pada elemen terakhir bernilai Null (\0).

Linked List

Linked List dapat juga digambarkan dalam susunan vertikal.



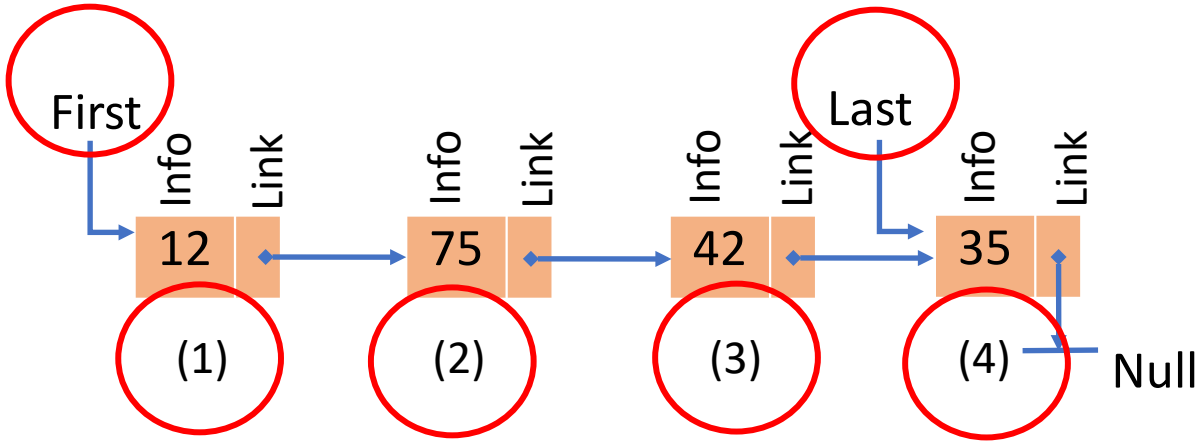
	Info	Link
1	I	6
2	M	1
3	O	8
4	L	9
5		
6	T	\0
7	N	3
8		4
9	I	2
10		

Start	=	7	Info[7]	=	N
Link(7)	=	3	Info[3]	=	O
Link(3)	=	8	Info[8]	=	spasi/ blank
Link(8)	=	4	Info[4]	=	L
Link(4)	=	9	Info[9]	=	I
Link(9)	=	2	Info[2]	=	M
Link(2)	=	1	Info[1]	=	I
Link(1)	=	6	Info[6]	=	T
Link(6)	=	Null			

*String pada contoh di atas adalah: **NO LIMIT**

4 macam struktur Linked List:

1) Linear Singly Linked List:

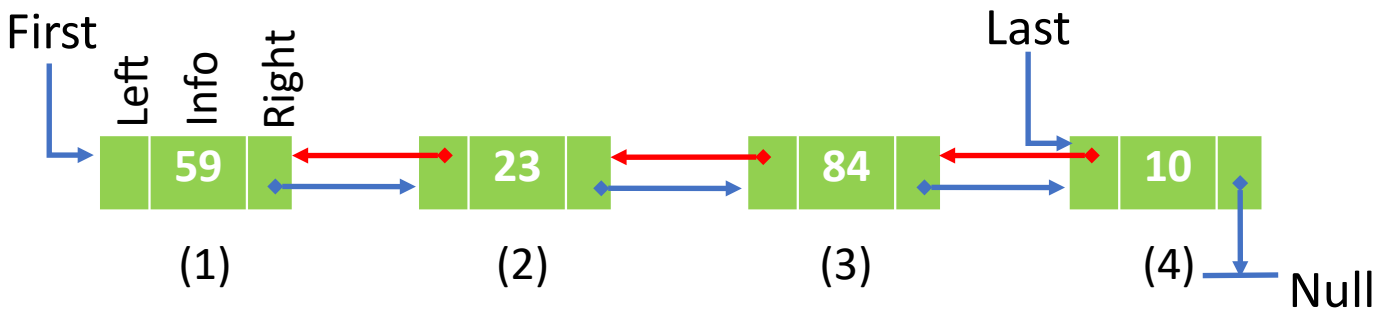


Contoh: Simpul (1)
Info(1) = 12 → Field Info(1) berisi nilai 12
Link(1) = 2 → Field Link(1) berisi alamat Simpul (2)

Keterangan:
ada 4 Simpul, Simpul nomor (1) s.d. (4).
Setiap Simpul (*record*) terdiri dari dua elemen (*field*), Field Info untuk menyimpan data dan Field Link bertipe pointer untuk menyimpan alamat Simpul berikutnya.
Simpul pertama ditunjuk oleh pointer *First* dan Simpul terakhir ditunjuk oleh pointer *Last*.

4 macam struktur Linked List:

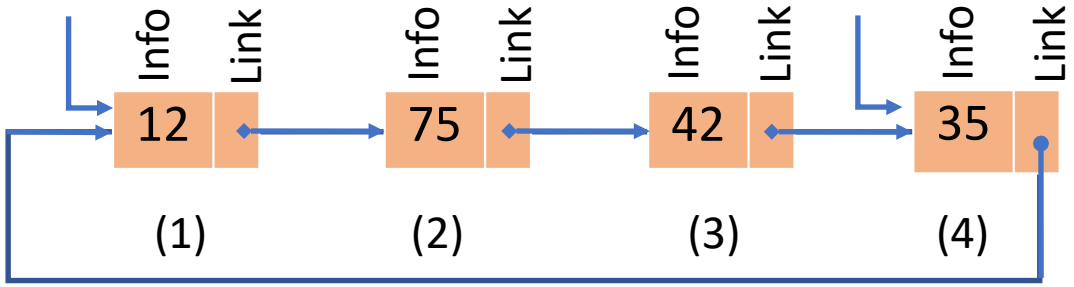
2) Linear Doubly Linked List:



Keterangan:
Setiap Simpul *(record)* memiliki dua pointer, pointer pertama (Left) menunjuk alamat simpul sebelumnya (previous node), sedang pointer kedua (Right) menunjuk alamat simpul berikutnya (*next node*).

4 macam struktur Linked List:

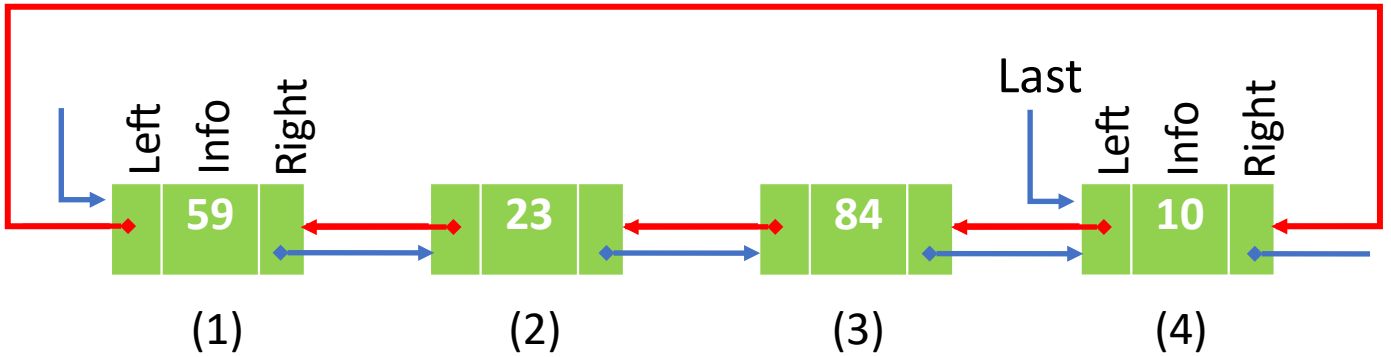
3) Circular Singly Linked List:



Keterangan:
Simpul terakhir berisi alamat simpul pertama, sehingga menghasilkan efek melingkar.

4 macam struktur Linked List:

4) Circular Doubly Linked List:



Keterangan:
Pointer Right pada Simpul Last berisi alamat Simpul First, sedangkan pointer Left pada Simpul First berisi alamat Simpul Last, sehingga menghasilkan efek melingkar dua arah yang berlawanan.

Kekurangan & Kelebihan:

Kekurangan:

- Diperlukan ruang tambahan untuk menyatakan tempat field pointer.
- Diperlukan waktu yang lebih banyak untuk mencari suatu Simpul/ Node dalam Linked List.

Kelebihan:

- Jenis data yang berbeda dapat dihubungkan (*Link*).
- Operasi Remove atau Insert dilakukan hanya dengan mengubah pointer nya saja.

KUNJUNGAN LINKED LIST

Traversal atau kunjungan simpul list sesuai urutan untuk memproses setiap simpul.
Algoritma traversal, menggunakan variabel PTR (pointer) untuk menunjuk simpul yang sedang di proses saat ini.
Statement **PTR := LINK(PTR)** ➔ akan menggerakkan pointer ke simpul berikutnya.

```
 Algoritma:
1.PTR := START.
2.Kerjakan Langkah 3&4 dalam hal PTR <> NULL :
3.Proses INFO(PTR).
4.PTR := LINK(PTR).
5.EXIT.
```

KUNJUNGAN LINKED LIST(lanjut.)

```
 Algoritma:
1.PTR := START.
2.Kerjakan Langkah 3&4 dalam hal PTR <> NULL :
3.Proses INFO(PTR).
4.PTR := LINK(PTR).
5.EXIT.
```

Penjelasan Algoritma: kita awali dengan memberi nilai kepada PTR, sama dengan START. Kita proses INFO(PTR), yakni informasi pada simpul pertama dalam List. Selanjutnya PTR diperbaharui melalui statement PTR := LINK(PTR). Sekarang proses INFO(PTR), yakni informasi pada simpul kedua. Demikian seterusnya sampai nilai PTR = NULL, akhir dari traversal.

PENCARIAN LINKED LIST

I. CARI DALAM LIST ACAK (TIDAK TERURUT) : melakukan Traversal Simpul list, sambil setiap kali memeriksa apakah informasi dalam simpul yang tengah dikunjungi tersebut sama dengan ITEM (data) yang dicari.

Algoritma:
SEARCH(INFO, LINK, START, ITEM, LOC)

1. PTR := START.
2. Kerjakan langkah 3 dalam hal PTR <> NULL :
3. Jika INFO(PTR) = ITEM, maka :
   LOC := PTR, exit.
   Bila tidak
   PTR := LINK(PTR).
4. LOC := NULL. (Pencarian gagal)
5. Exit.

PENCARIAN LINKED LIST(lanjut.)

Algoritma:
SEARCH(INFO, LINK, START, ITEM, LOC)

1. PTR := START.
2. Kerjakan langkah 3 dalam hal PTR <> NULL :
3. Jika INFO(PTR) = ITEM, maka :
   LOC := PTR, exit.
   Bila tidak
   PTR := LINK(PTR).
4. LOC := NULL. (Pencarian gagal)
5. Exit.

Dalam algoritma ini diperlukan 2 buah pemeriksaan pada setiap putaran yaitu :

1. Memeriksa apakah telah sampai pada akhir dari list, yakni dengan memeriksa apakah PTR = NULL.
2. Memeriksa apakah ITEM telah ditemukan lokasinya, yakni dengan memeriksa apakah INFO(PTR) = ITEM.

PENCARIAN LINKED LIST(lanjut.)

II. CARI DALAM LIST TERURUT : melakukan Traversal Simpul list, sambil setiap kali memeriksa apakah informasi dalam simpul yang tengah dikunjungi tersebut sama dengan ITEM yang dicari. Karena terurutnya list, tidak perlu melakukan Traversal sampai akhir dari list, walau ITEM tidak terdapat dalam list.

Begitu INFO(PTR) > ITEM, hentikan proses pencarian.

Agoritma:
SRCHSL(INFO, LINK, START, ITEM, LOC)

1. PTR := START.
2. Kerjakan langkah 3 dalam hal PTR <> NULL :
3. Jika INFO(PTR) < ITEM, maka :
   PTR := LINK(PTR).
   Bila tidak jika ITEM = INFO(PTR), maka :
   LOC := PTR, dan exit. (pencarian sukses)
   Bila tidak :
   LOC := NULL, and exit.
4. LOC := NULL.
5. Exit.

PENCARIAN LINKED LIST(lanjut.)

Agoritma:
SRCHSL(INFO, LINK, START, ITEM, LOC)

1. PTR := START.
2. Kerjakan langkah 3 dalam hal PTR <> NULL :
3. Jika INFO(PTR) < ITEM, maka :
   PTR := LINK(PTR).
   Bila tidak jika ITEM = INFO(PTR), maka :
   LOC := PTR, dan exit. (pencarian sukses)
   Bila tidak :
   LOC := NULL, and exit.
4. LOC := NULL.
5. Exit.

Dalam algoritma ini diperlukan 2 buah pemeriksaan pada setiap putaran yaitu :

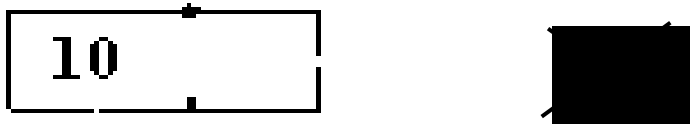
1. Memeriksa apakah INFO(PTR) sudah lebih besar dari ITEM, berarti ITEM tidak terdapat dalam list, hentikan proses cari.
2. Memeriksa apakah ITEM telah ditemukan lokasinya, yakni dengan memeriksa apakah INFO(PTR) = ITEM.

Operasi Linked List

1. List masih kosong (head=NULL)

head

2. Masukkan data baru, misal 10



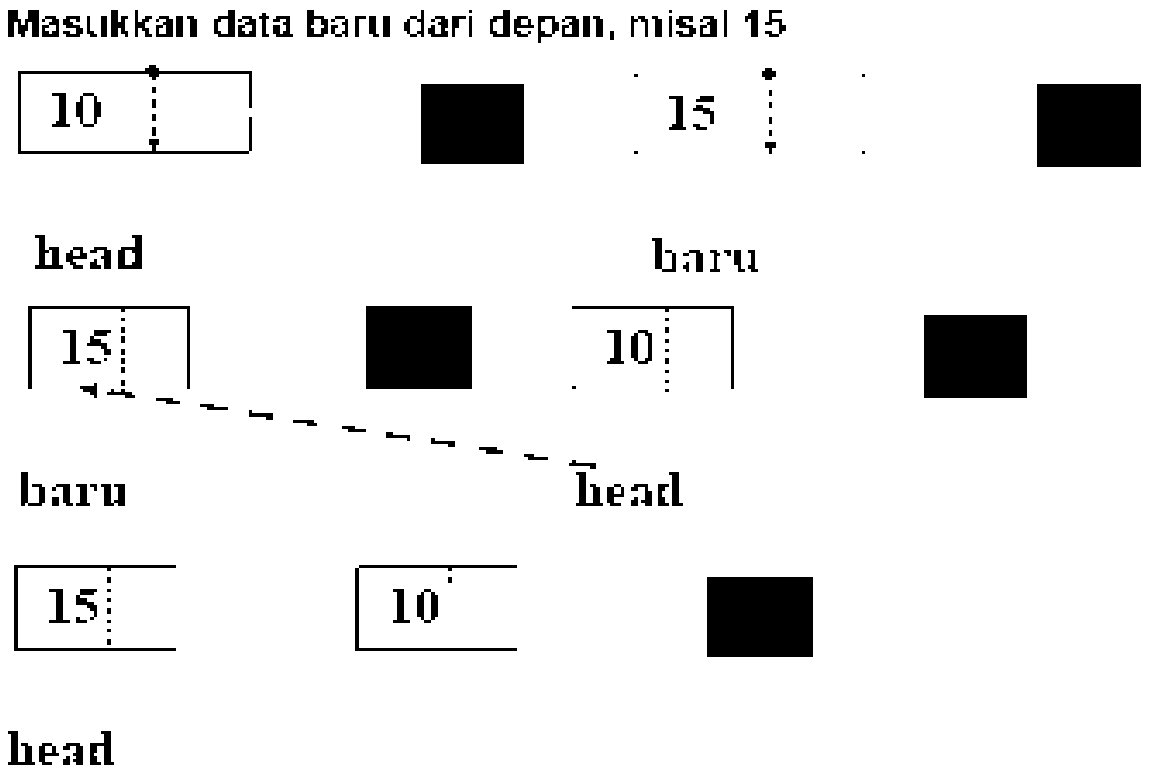
head

Penambahan elemen di posisi awal: data baru akan menjadi awal.

Ada 2 hal yang harus diperhatikan, yaitu :

1. Kondisi Linked List sedang kosong ➔ variable awal dan akhir akan diisi dengan variable baru;/
2. Kondisi Linked List sudah mempunyai elemen ➔ mengisi field next milik elemen baru dengan posisi di awal Linked List, lalu posisi awal berubah ke posisi baru.

Operasi Linked List (lanjut.)

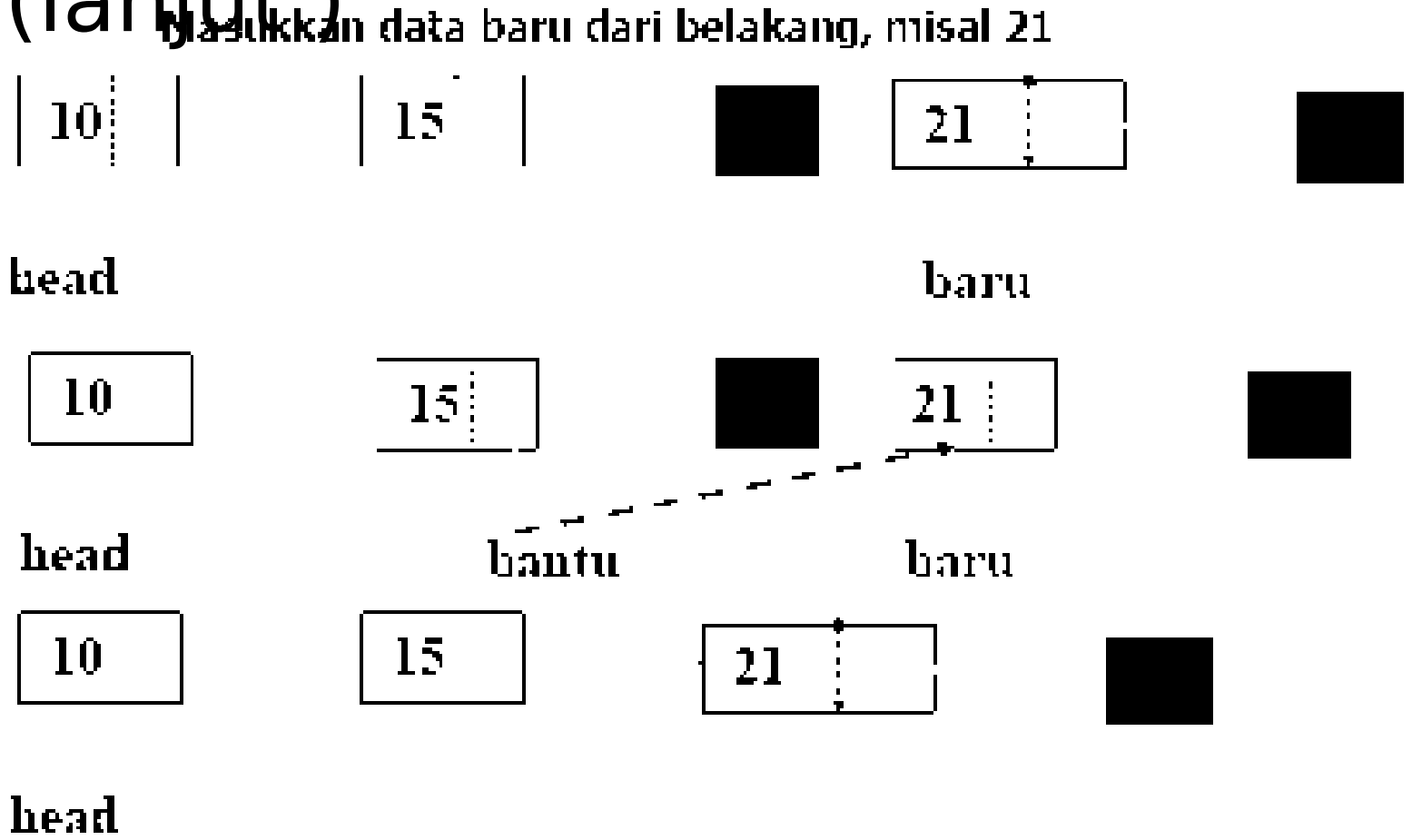


Penambahan elemen di posisi awal: data baru akan menjadi awal.

Ada 2 hal yang harus diperhatikan, yaitu :

1. Kondisi Linked List sedang kosong ➔ variable awal dan akhir akan diisi dengan variable baru; atau
2. Kondisi Linked List sudah mempunyai elemen ➔ mengisi field next milik elemen baru dengan posisi di awal Linked List, lalu posisi awal berubah ke posisi baru.

Operasi Linked List (lanjut..)



Penambahan elemen di posisi terakhir: setelah proses penambahan selesai, maka variabel menunjuk ke data baru tersebut.

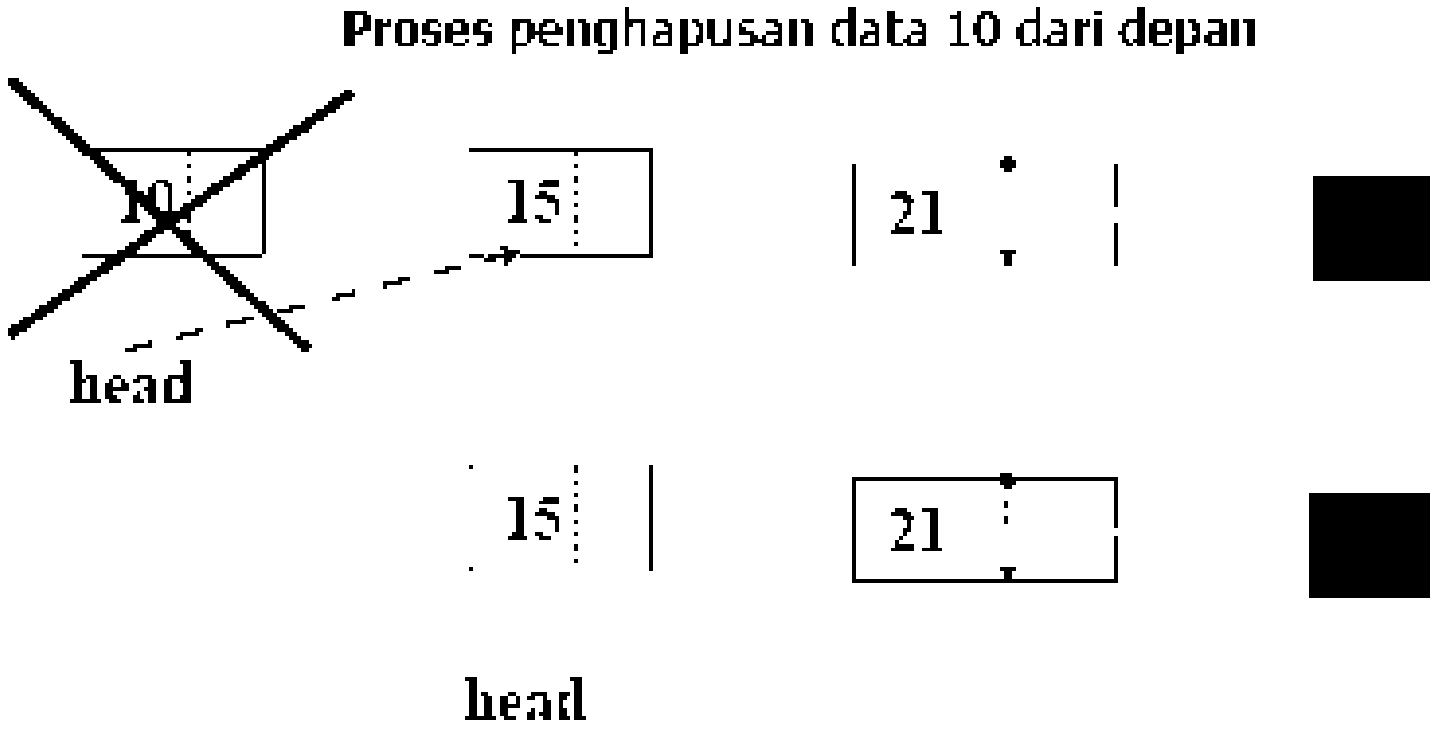
Operasi linked list (lanjut.)

Menghapus elemen list: *menghilangkan alokasi memori sebuah List yang telah ada di memori.*

Fungsi: agar data yang tidak diperlukan benar-benar terhapus di memori sehingga penggunaan memori dapat optimal karena data-data yang tidak diperlukan dihilangkan.

- *Penghapusan Simpul/ Node tidak bisa dilakukan jika keadaan Simpul sedang ditunjuk oleh Pointer.*
- *Sebelum data terdepan dihapus, Head harus ditunjukkan ke Simpul berikutnya terlebih dahulu agar List tidak putus, sehingga Simpul tersebut akan menjadi Head baru (data terdepan yang baru).*

Operasi linked list (lanjut.)



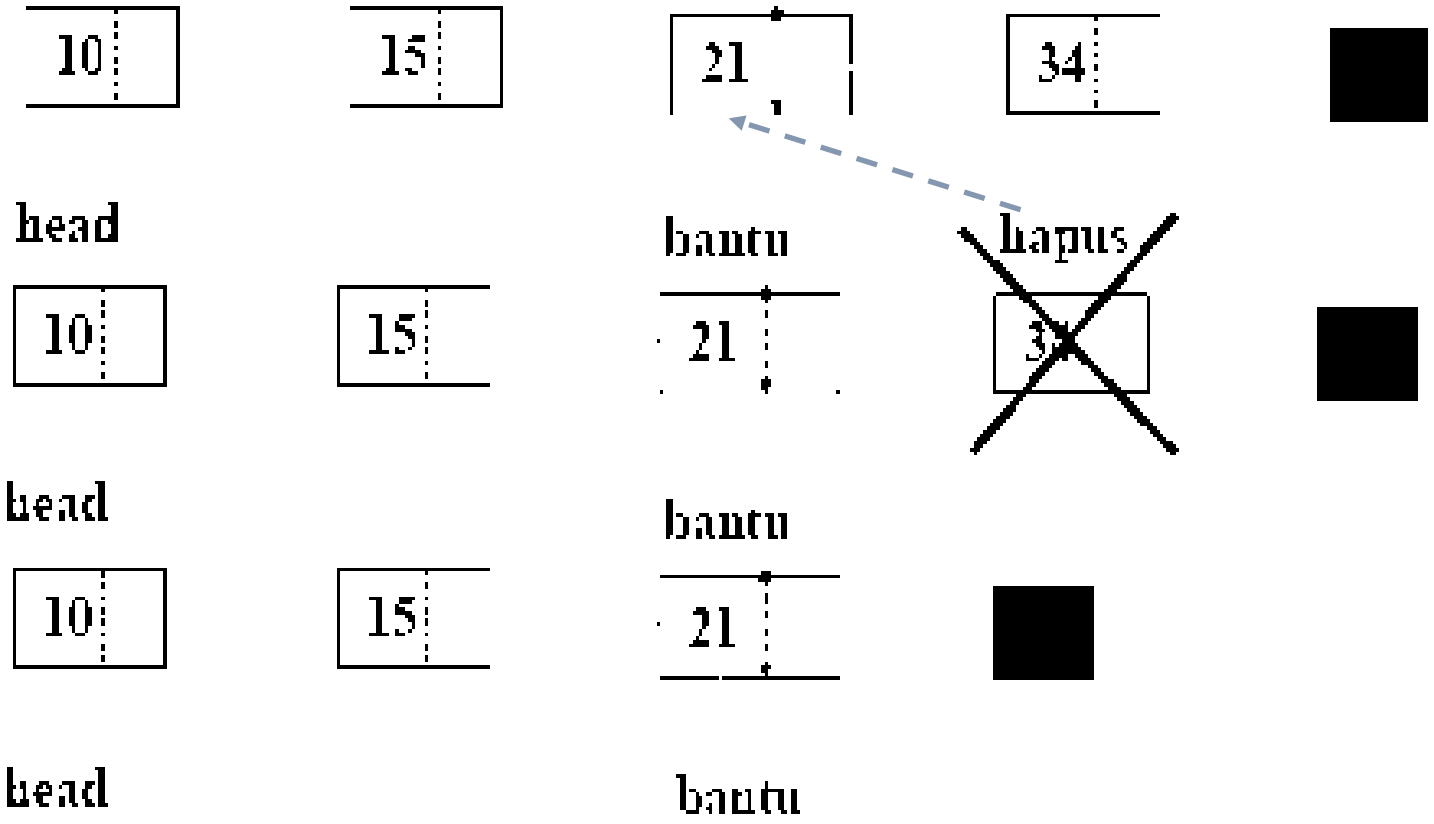
Penghapusan elemen pertama (awal), menyebabkan variabel awal akan berpindah ke elemen data berikutnya.

Kondisi yang perlu diperhatikan yaitu:

Kondisi Linked List masih kosong proses tidak bisa dilakukan

Operasi linked list (lanjut.)

Proses menghapus data 34 dari belakang



Penghapusan data akhir adalah proses menghilangkan/ menghapus data yang ada di posisi terakhir.

Koleksi Sampah (Garbage Collection)

Ketika menyimpan Linked List dalam memori, diasumsikan bahwa selalu dapat dilakukan penyisipan Simpul baru ke dalam List, serta penghapusan Simpul dari List.

Untuk itu diperlukan suatu mekanisme guna menyediakan memori bagi Simpul baru, atau untuk mengelola memori yang sementara ini tidak berguna karena adanya penghapusan Simpul, untuk sewaktu-waktu dapat dipakai lagi.

Sel memori dalam array yang tak digunakan, dihimpun menjadi sebuah Linked List lain yang menggunakan variabel pointer List berupa array AVAIL, dituliskan sebagai : LIST(INFO, LINK, START, AVAIL)

Koleksi Sampah (Garbage Collection)

Koleksi sampah (*Garbage Collection*) adalah suatu metode dengan sistem operasi yang dapat secara periodik mengumpulkan semua ruang kosong akibat penghapusan Simpul List ke dalam List ruang bebas.

Koleksi Sampah (Garbage Collection)

	Info	Link
1	I	6
2	M	1
3	O	8
4	L	9
5		\0
6	T	\0
7	N	3
8		4
9	I	2
10		5

Start → 7

Avail → 10

Sel memori dalam array yang tak digunakan, dihimpun menjadi sebuah Linked List lain yang menggunakan variabel Pointer List berupa array AVAIL, dituliskan sebagai :

LIST(INFO, LINK, START, AVAIL)

Avail = 10, maka Info(10) adalah Simpul bebas pertama dalam List Avail tersebut. Karena Link(Avail) = LINK(10) = 5, maka Info(5) adalah simpul bebas kedua dalam Avail dan juga simpul terakhir yang kosong.


TERIMA KASIH
FAIR &
RESPONSIBLE