Linked List

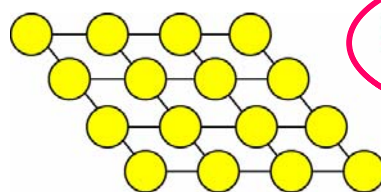
Review : Sifat Implementasi Linear List dengan Array

- Pencarian dapat dilakukan dengan mudah.
- Melihat elemen dalam list dapat dilakukan dengan mudah
- Membaca elemen dalam list dapat dilakukan dengan mudah
- Proses menyisipkan dan menghapus elemen tidak mudah. Mengapa ?
  - ✓ Elemen-elemen harus digeser
- Sulit untuk memperkirakan jumlah array yang harus dibuat.

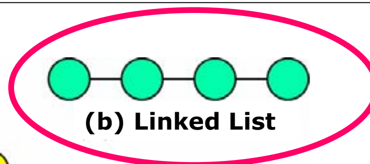
Review : Sifat Implementasi Linear List dengan Array

- Bagaimana mengatasi kelemahan implementasi linear list dengan array?
  - Satu elemen dengan elemen yang lain dihubungkan dengan "suatu penghubung (link)"
- Lahir gagasan linked list

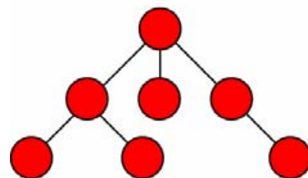
Beberapa Struktur Data



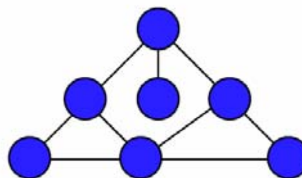
(a) Matriks



(b) Linked List



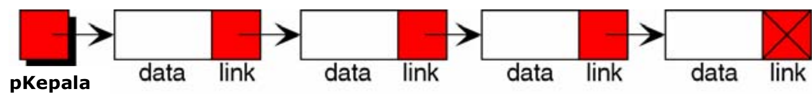
(c) Tree



(d) Jaringan

Implementasi *linked list* (list terhubung)

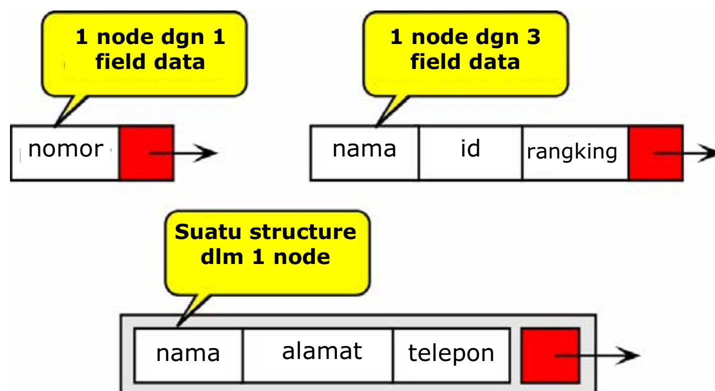
- *Linked list* merupakan rantai elemen
- Tiap elemen punya :
  - data
  - link yang menunjuk ke elemen berikut



(a) Suatu linked list dengan pointer kepala: pKepala

(b) Suatu linked list kosong

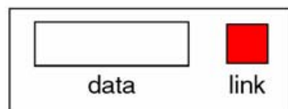
Node : Suatu elemen dari Linked List



Kepala & Node menggunakan Struktur Data Structure

- Kita perlu mendefinisikan 2 structure:
  - satu untuk kepala (head) list
  - satu untuk node list

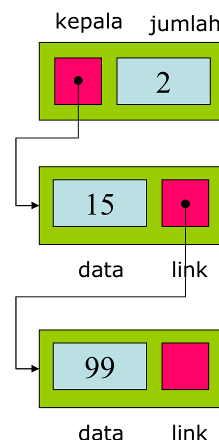
(a) Structure utk kepala



(b) Structure utk node

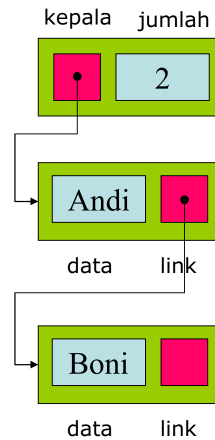
Kepala dan Node : list integer sederhana

```
struct intlist {
    struct node *kepala;
    int jumlah;
}IntList;

struct node {
    int data;
    struct node *link;
}Node;
```



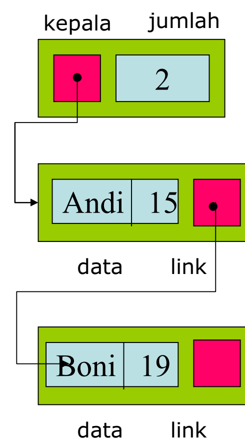
Kepala dan Node: list nama

```
struct IntList {
    struct node *kepala;
    int jumlah;
}IntList;

struct node {
    char *name;
    struct node *link;
}Node;
```



Kepala dan Node: structure sebagai data

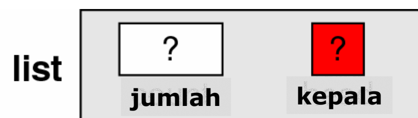
```
struct Intlist {
    struct node *kepala;
    int jumlah;
};

struct node {
    struct person *data;
    struct node *link
}Node;

struct person {
   char *nama;
   int umur;
}
```



Operasi-Operasi Utama dalam Ordered Linked List

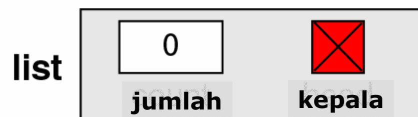
- Buat List
- Tambah node
  - Di awal, tengah atau akhir
- Hapus node
  - Di awal, tengah atau akhir
- Cari node
- Melacak list (Membaca node-node)

Operasi (1) Buat list



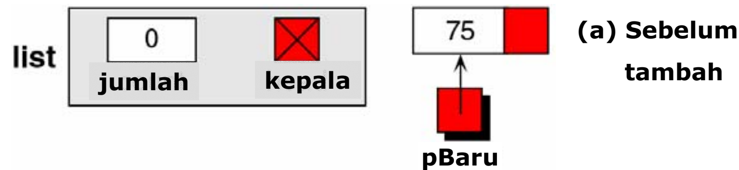
(a) Sebelum buat

```
list.kepala = null
list.jumlah = 0
```

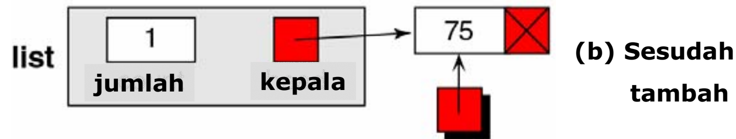


(b) Setelah buat

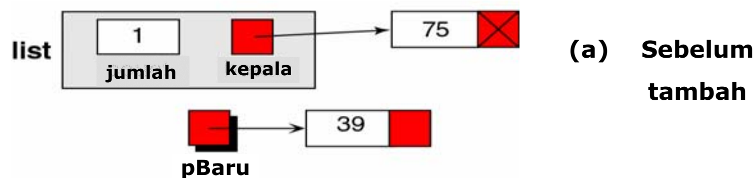
Operasi (2) Tambah Node : Jika List Kosong



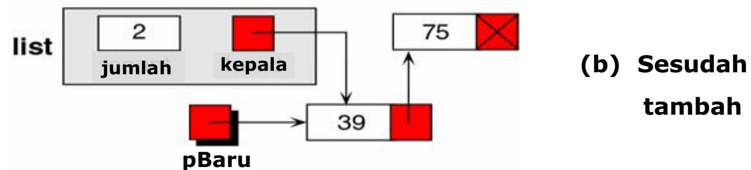

```
pBaru -> link = list.kepala
list.kepala = pBaru
```



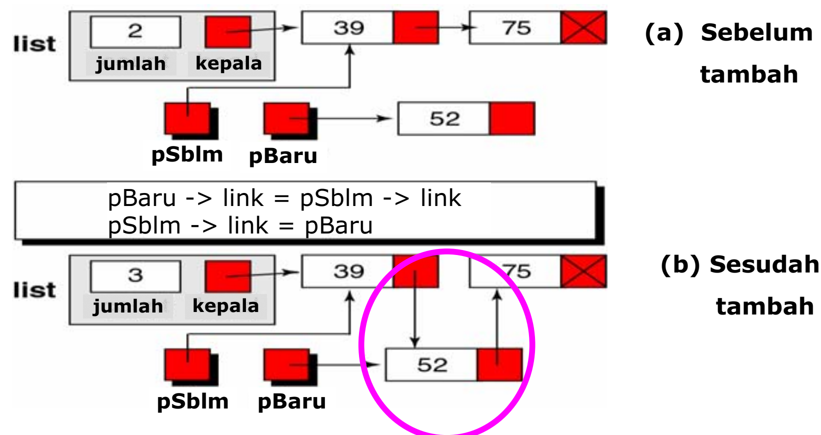
Operasi Tambah Node: Di Awal List




```
pBaru -> link = list.kepala
list.kepala = pBaru
```



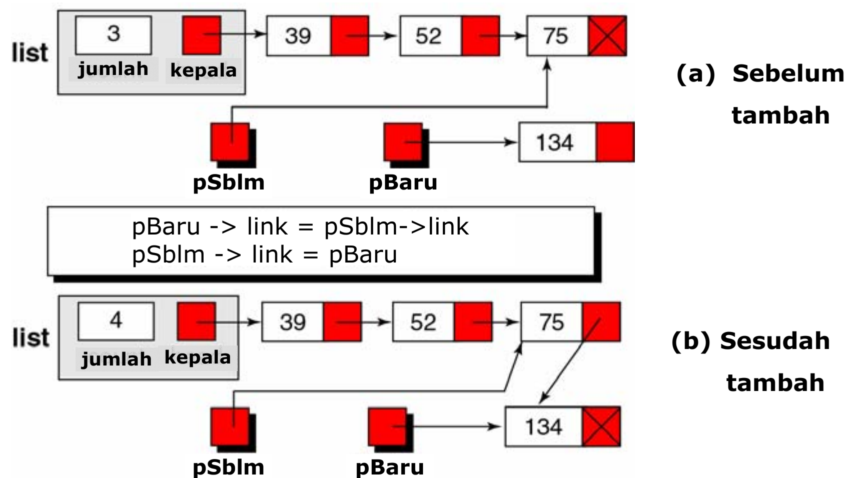
Operasi Tambah Node : Di Tengah List



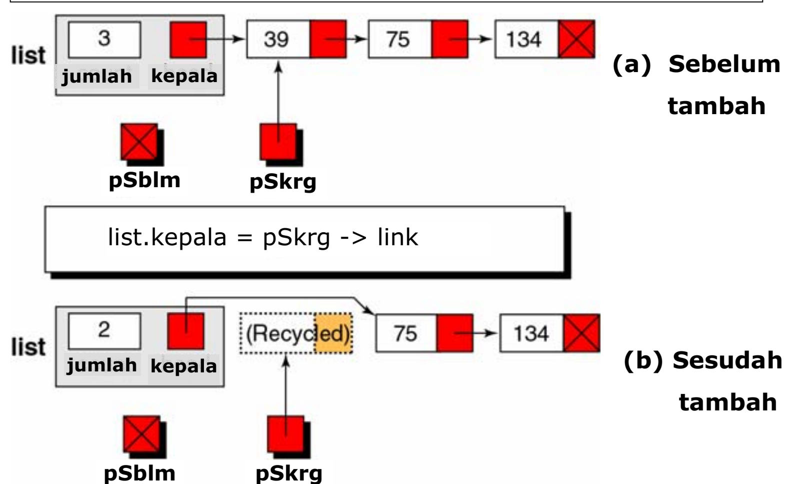
Operasi Tambah node: Di Tengah List

- Bandingkan operasi penambahan elemen dalam implementasi linear list dengan array dan dengan linked list
- Apa kesimpulan Anda?

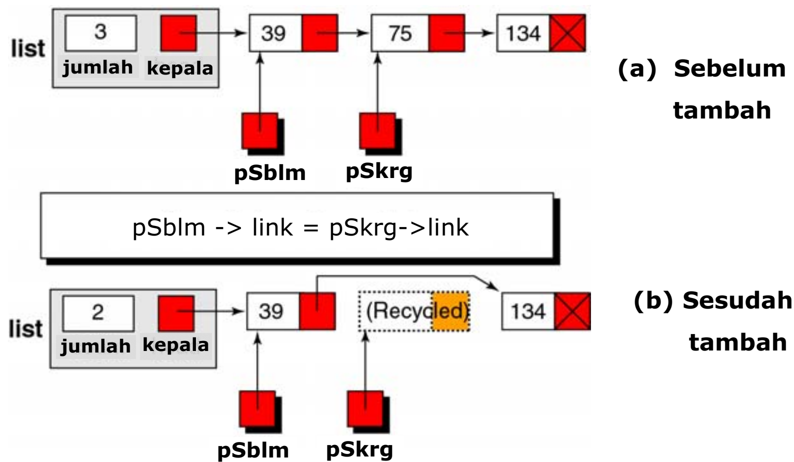
Operasi Tambah Node : Di Akhir List



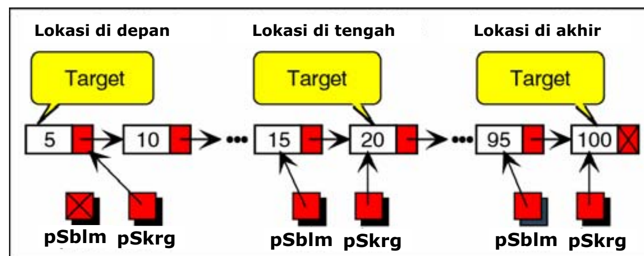
Operasi (3) Hapus Node: Di Awal List



Operasi Hapus Node : Kasus Umum

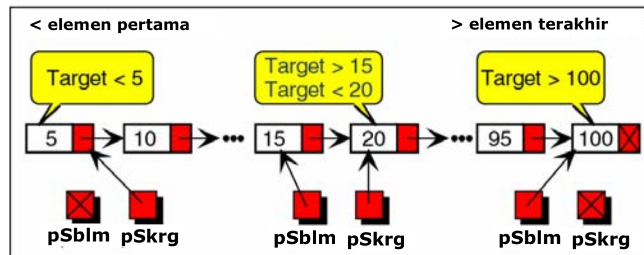


Operasi (4) Pencarian node



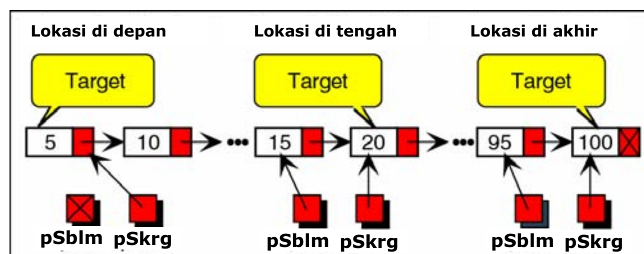
(a) Pencarian sukses (return true)

Operasi (4) Pencarian node



(b) Pencarian gagal (return false)

Operasi Melacak List



Dilakukan proses pelacakan / pembacaan untuk setiap node

Mengapa harus dengan pointer?

- Pada beberapa bahasa pemrograman yang tidak memiliki tipe data pointer, linked list diimplementasikan dengan array

Struktur Data : Implementasi Linked Lists Dengan Variabel Dinamis

Alokasi Memori Dinamis

- Alokasi memori dinamis
  - Memakai dan membebaskan memory selama proses eksekusi
- **malloc**
  - Mengalokasikan sejumlah byte untuk digunakan
    - **sizeof** digunakan untuk menentukan ukuran obyek
  - Akan mengembalikan pointer ke sebuah lokasi di memori.
    - Jika tidak ada memori yang tersedia akan mengembalikan **NULL**
  - **P = (Node *)malloc(sizeof(struct Node));**
    - P akan menunjuk ke variabel dinamis baru bertipe node yang telah dibuat.

Alokasi Memori Dinamis

- **free**
  - Membebaskan memori yang dibuat dengan malloc yang sudah tidak digunakan
  - **free (P);**