Linked List

BAB 2

Materi

2.1 Linked List

2.2 Single Linked List

- 2.2.1 Representasi Simpul (Node)
- 2.2.2 Alokasi Simpul
- 2.2.3 Operasi pada Linked List

2.3 Double Linked List

2.4 Circular List

2.5 Kesimpulan

2.1 Linked List

Pengolahan data yang dilakukan dalam sebuah computer terdiri dari masukan, penyimpanan dan pengolahan data.

Penyimpanan data dan pengolahan lain dari sekelompok data yang telah terorganisir dalam sebuah urutan tertentu.

Salah satu cara untuk menyimpan sekumpulan data yang kita miliki adalah menggunakan larik atau Array.

Link List adalah sebuah data atau sekumpulan data yang masing-masing dihubungkan dengan pointer tunggal.

2.1 Linked List

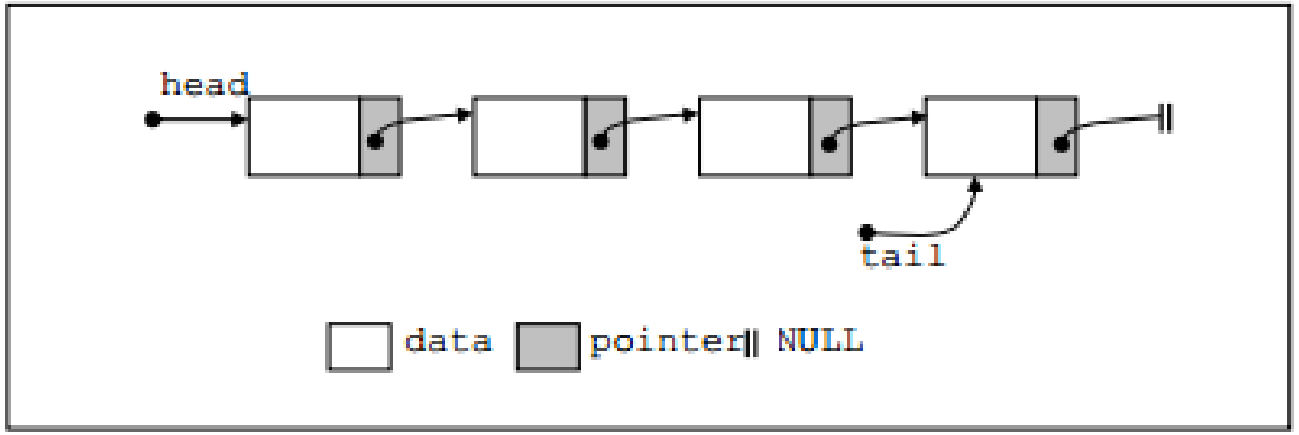
Masing-masing elemen terdiri dari dua bagian

1. adalah data itu sendiri

2. pointer dari data tersebut

2.2 Single Linked List

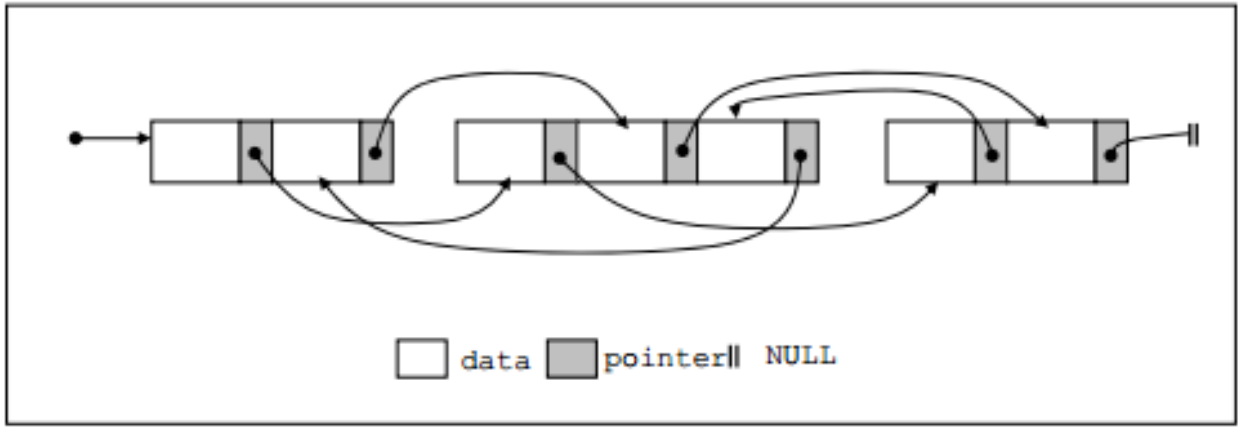
Dengan menggunakan struktur two-member seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya.



Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head, dan elemen terakhir dari suatu list disebut tail.

Secara konseptual, linked list merupakan deretan elemen yang berdampingan.

Akan tetapi, karena elemen-elemen tersebut dialokasikan secara dinamis (menggunakan malloc), sangat penting untuk diingat bahwa kenyataannya, linked list akan terpencarpencar di memori.



Pointer dari elemen ke elemen berarti sebagai penjamin bahwa semua elemen dapat diakses.

2.2.1 Representasi Simpul (Node)

Struktur node pada linked list merupakan suatu simpul(node) yang berisi pointer ke suatu data yang merupakan data dirinya sendiri.

```
typedef struct node *list;
struct node {
int datalist;
struct node *next;
};
```

dilanjutkan dengan deklarasi dari pointer ke struktur di atas sebagai berikut:

```
struct node *head;
```

atau

```
list head;
```

2.2.2 Alokasi Simpul

Ketika sebuah variabel dideklarasikan, terlebih dahulu harus diinisialisasi. Demikian juga dengan pengalokasian secara dinamis.

Sehingga, fungsi untuk mengalokasikan sebuah node baru, fungsi *allocate_node()* menggunakan *malloc()* untuk mendapatkan memori aktual, yang akan menginisialisasi suatu field data. *next* selalu diinisialisasi sebagai NULL.

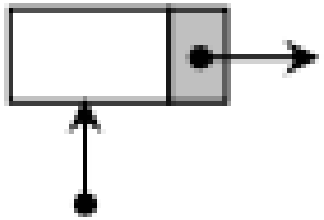
```
int allocate_node(int data, list *new)
{
        new = (list) malloc (sizeof(node));
        if(new==NULL)
        return 0;
        new->datalist = data;
        new->next=NULL;
        return 1;
}
```

Inisialisasi List

Untuk inisialisasi list setelah alokasi untuk node pertama maka ditambahkan statement sebagai berikut:

head = new;

Ilustrasi dari fungsi i allocate_node() adalah sebagai berikut

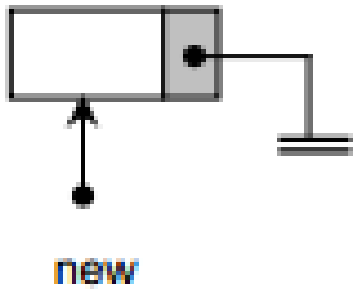


new = (list) malloc (sizeof(node));

new->datalist = data;

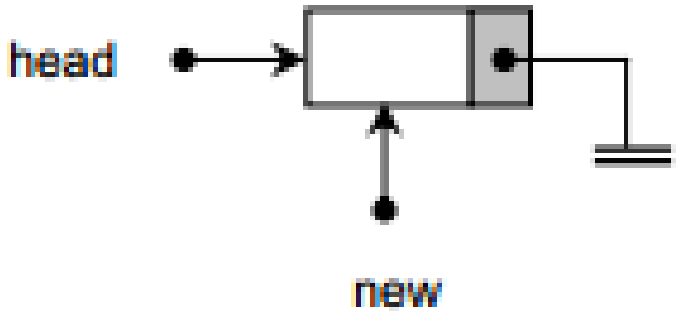
Inisialisasi List

new->next=NULL;



Untuk inisialisasi list setelah alokasi untuk node pertama ilustrasinya adalah sebagai berikut:

head = new;



Fungsi Free_node()

Fungsi free_node()menggunakan memori dinamis dengan fungsi free().

free() akan menghancurkan node yang menunjuk ke pointer yang dilewati, sehingga tempat yang ada dapat dipakai untuk lainnya.

Akan tetapi, pointer yang melewati free() tidak otomatis menjadi null, tetapi akan menunjuk ke node yang sudah tidak ada.

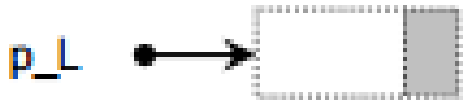
Karena itu pointer harus didefinisikan dengan NULL.

Penggunaan free_node()

```
void free_node(list p_L)
{
        free(p_L);
        p_L=NULL;
}
```

Ilustrasi dari fungsi free_node()

free(*p_L);



*p_L=NULL;

2.2.3 Operasi pada Linked List

1. Insert
    1. Insert pada node awal (head) dari linked list
    2. Insert setelah node tertentu
    3. Insert sebelum node tertentu
    4. Insert pada akhir (tail) dari linked list
2. Delete
    1. Penghapusan Simpul Pertama
    2. Penghapusan Setelah Simpul Tertentu
    3. Penghapusan Simpul Terakhir

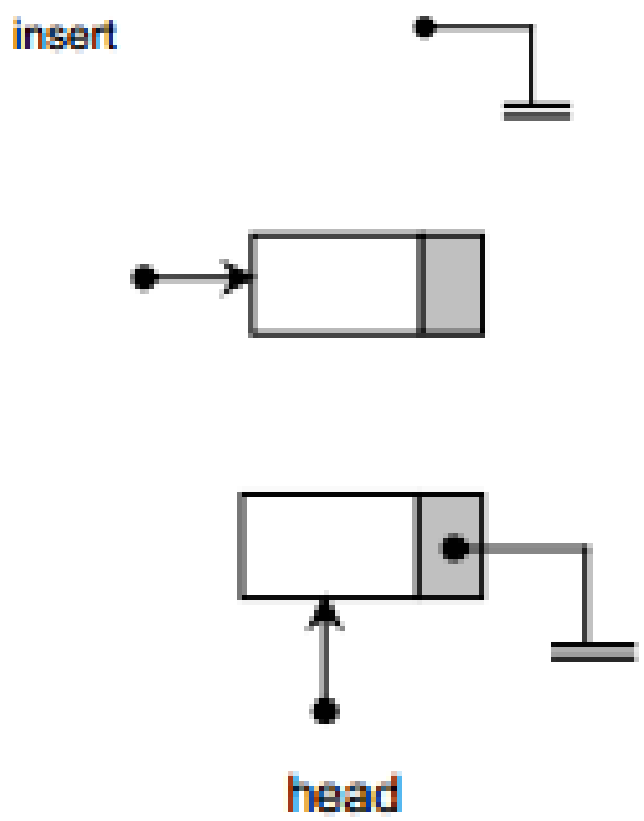
2.2.3.1 Insert

1. Insert pada node awal (head) dari linked list
2. Insert setelah node tertentu
3. Insert sebelum node tertentu
4. Insert pada akhir (tail) dari linked list

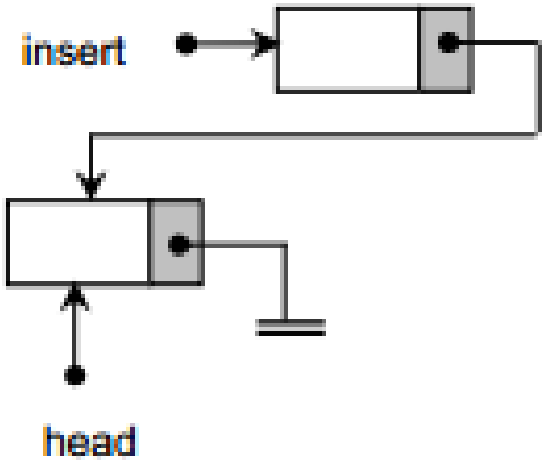
Insert pada node awal (head) dari linked list

```
void insertashead(list insert)
{
insert->next=head;
head = insert;
}
```

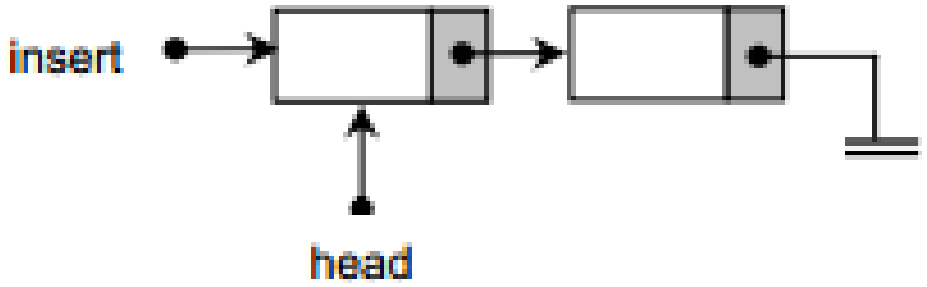
Ilustrasi :



insert->next=head;



head = insert;

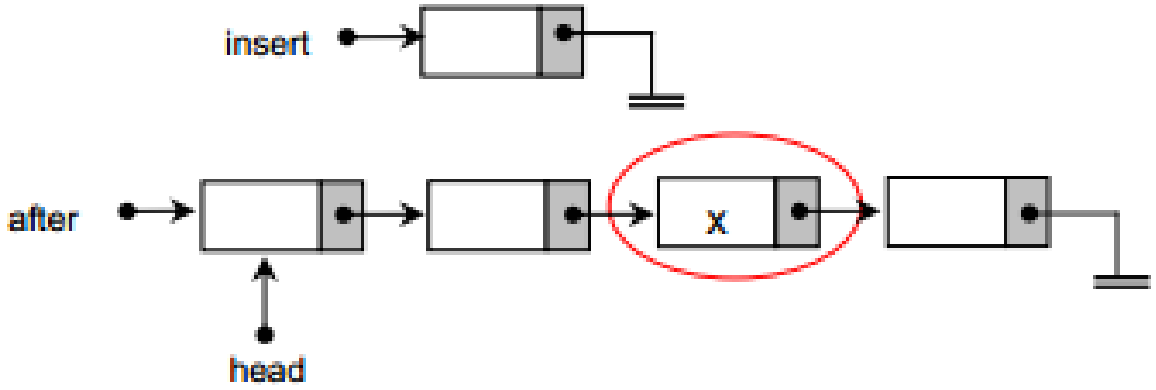


Insert setelah node tertentu

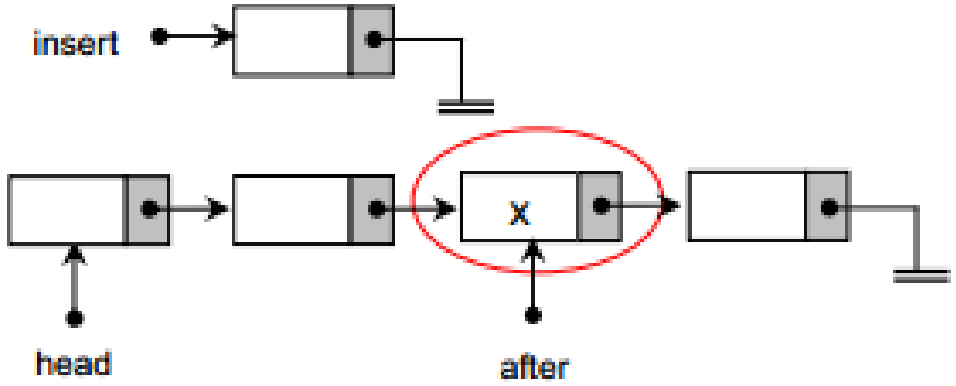
```
void insertafternode(int x, list insert)
{
list after;
after = head;
do
after = after->next;
while (after->datalist != x);
insert->next = after->next;
after->next = insert;
}
```

Insert setelah node tertentu (ilustrasi)

after = head;

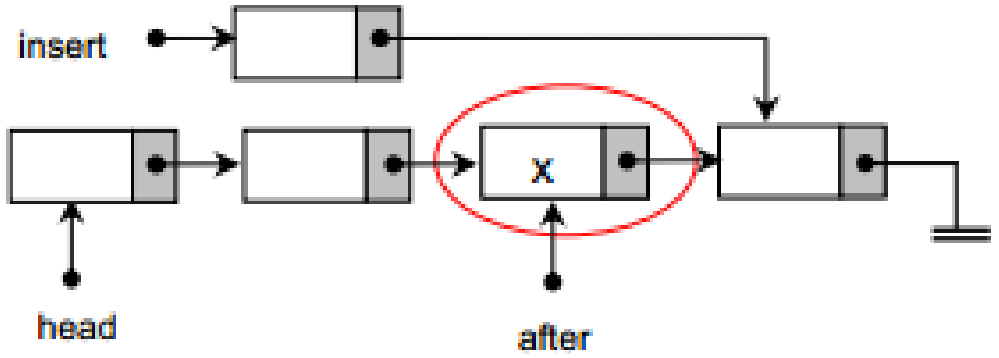


do
after = after->next;
while (after->datalist != x);

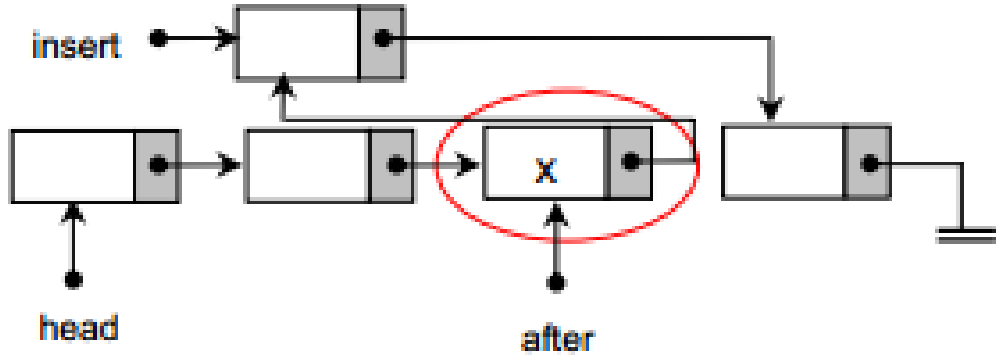


Insert setelah node tertentu (ilustrasi)

insert->next = after->next;



after->next = insert;

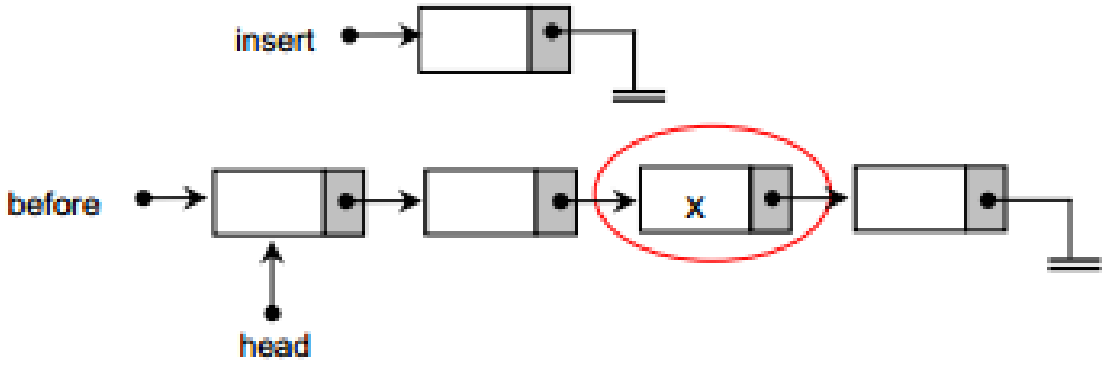


Insert sebelum node tertentu

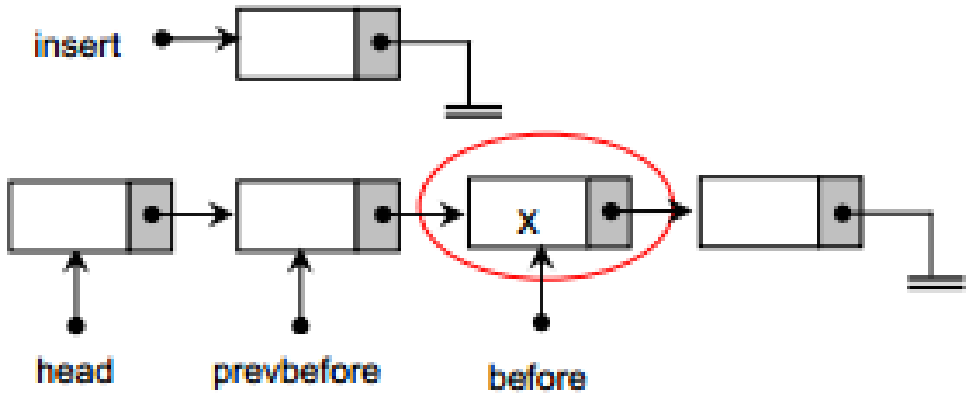
```
void insertbeforenode(int x, list insert)
{
        list before, prevbefore;
        if (head->datalist = x)
        insertashead(insert)
        else
{
        before = head;
        do
        prevbefore = before;
        before = before->next;
        while (before->datalist != x);
        insert->next = before;
        prevbefore->next = insert;
        }
}
```

Insert sebelum node tertentu (ilustrasi)

before = head;

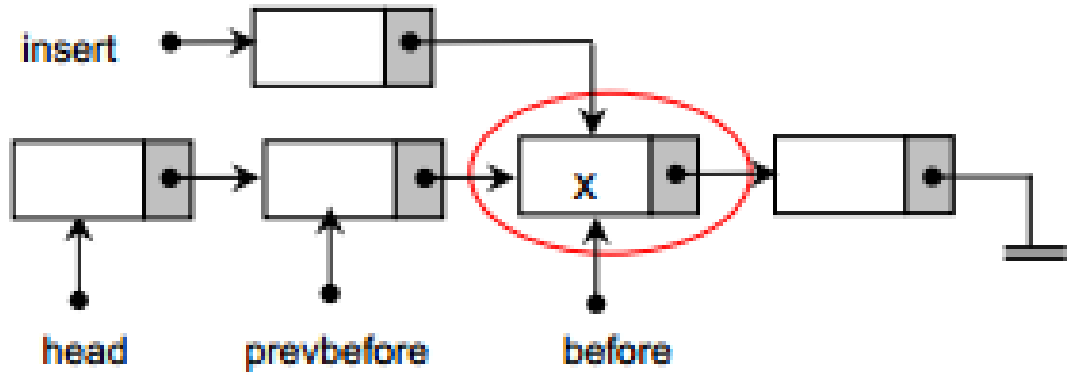


do
prevbefore = before;
before = before->next;
while (before->datalist != x);

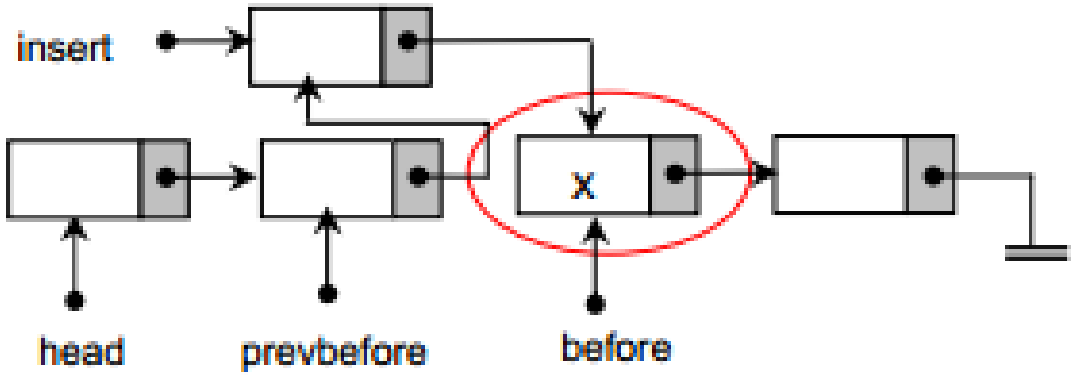


Insert sebelum node tertentu (ilustrasi)

insert->next = before;



prevbefore->next = insert;

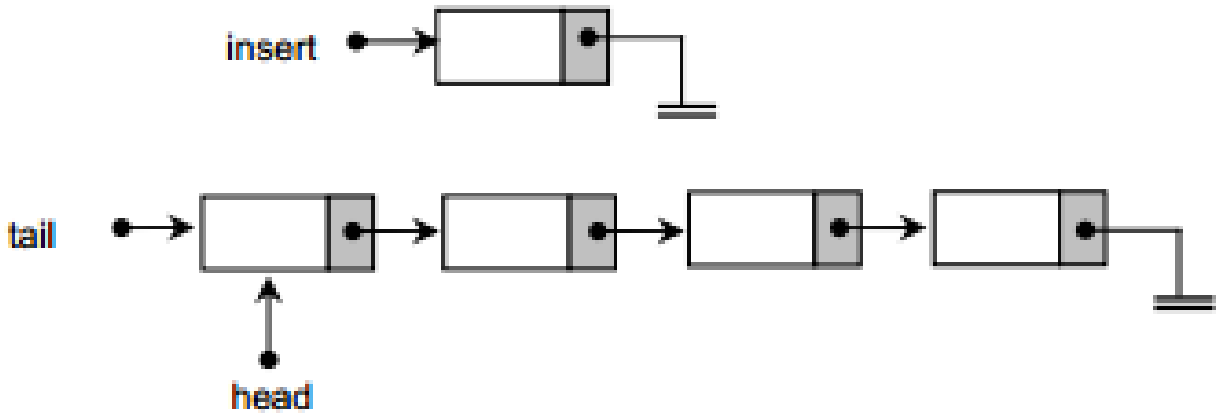


Insert pada akhir (tail) dari linked list

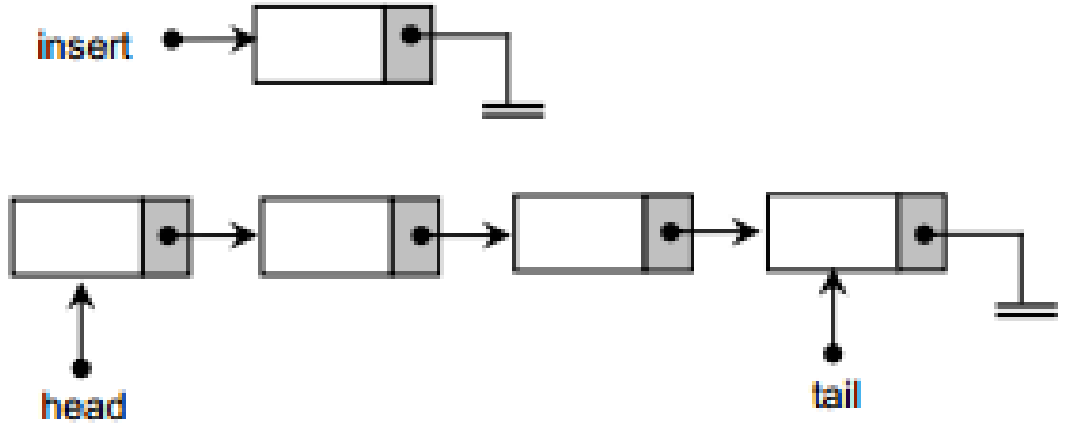
```
void insertastail(list insert)
{
list tail;
tail = head;
do
tail = tail->next;
while (tail->next != NULL);
tail->next = insert;
tail = tail->next;
}
```

Insert pada akhir (tail) dari linked list (ilustrasi)

tail = head;

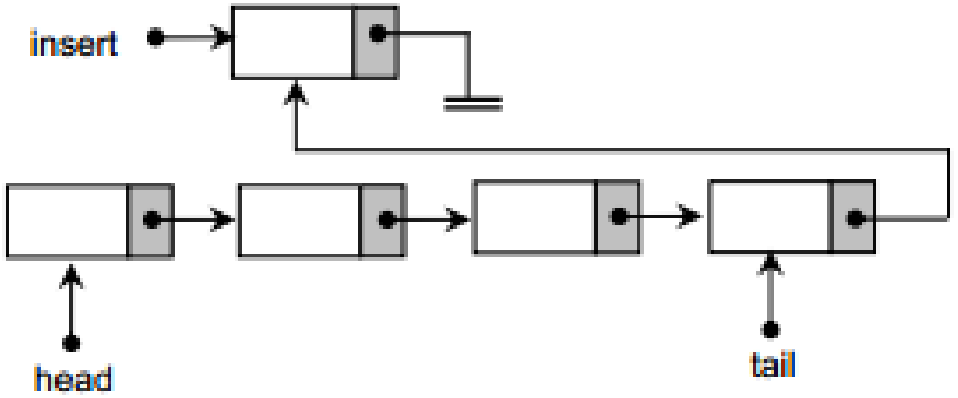


do
tail = tail->next;
while (tail->next != NULL);

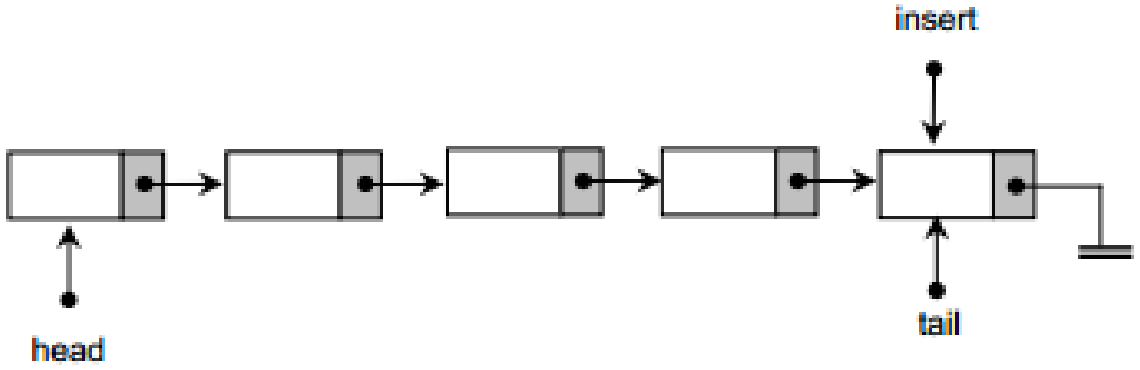


Insert pada akhir (tail) dari linked list (ilustrasi)

tail->next = insert;



tail = tail->next;



2.2.3.2 Delete

1. Penghapusan Simpul Pertama
2. Penghapusan Setelah Simpul Tertentu
3. Penghapusan Simpul Terakhir

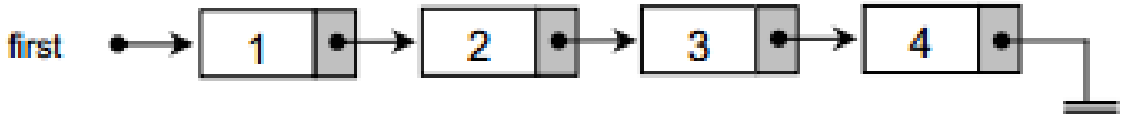
Penghapusan Simpul Pertama

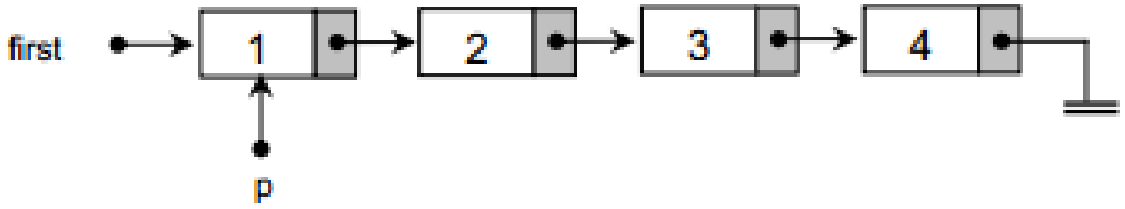
Langkah-langkah untuk proses di atas adalah sebagai berikut:

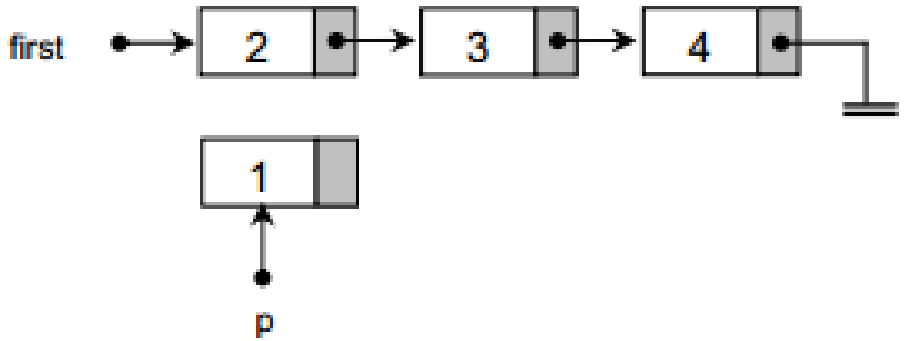
1. Pointer first diarahkan pada data ke-2

2. Pointer p diarahkan pada data ke-1

3. Bebaskan pointer p (secara otomatis data ke-1 terhapus)


first
1
2
3
4


first
1
2
3
4
p

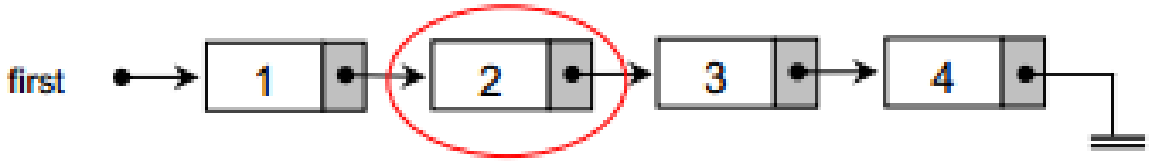

first
2
3
4
1
p

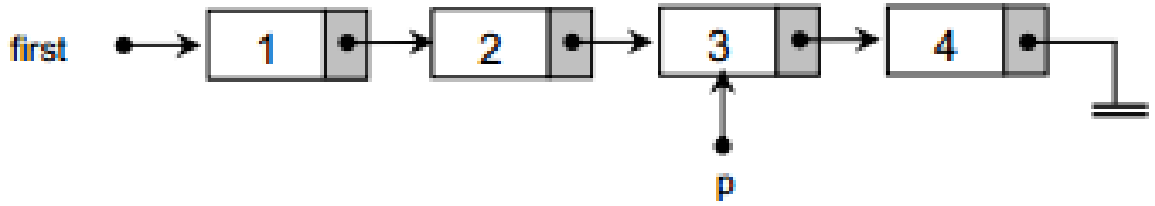

first
2
3
4

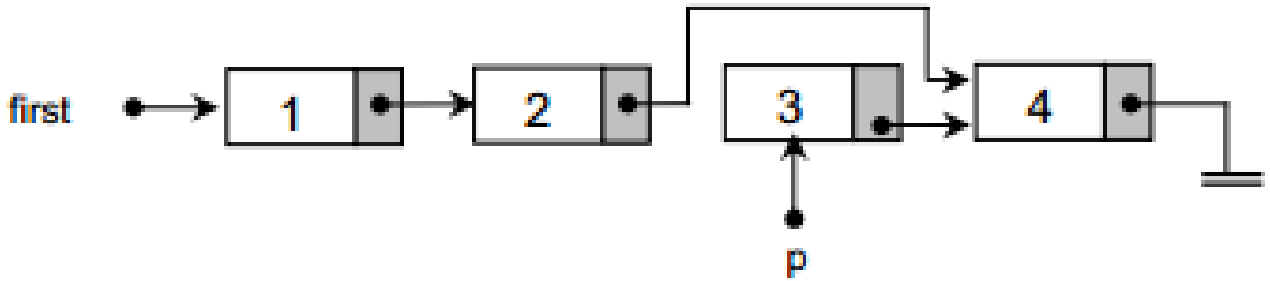
Penghapusan Setelah Simpul Tertentu

Langkah-langkah untuk proses di atas adalah sebagai berikut:

1. Arahkan pointer first s/d data yang ditunjuk
2. Pointer p diarahkan pada first->next
3. Arahkan pointer first->next pada p->next
4. Bebaskan pointer p (secara otomatis data setelah simpul tertentu terhapus)


first
1
2
3
4


first
1
2
3
4
p


first
1
2
3
4
p


first
1
2
4

Penghapusan Simpul Terakhir

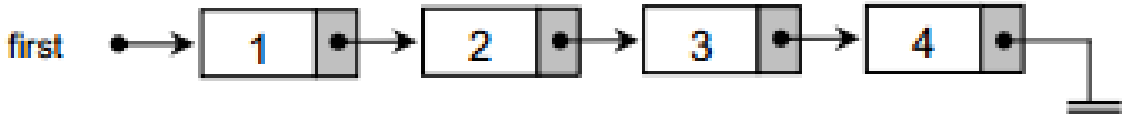
Langkah-langkah untuk proses di atas adalah sebagai berikut:

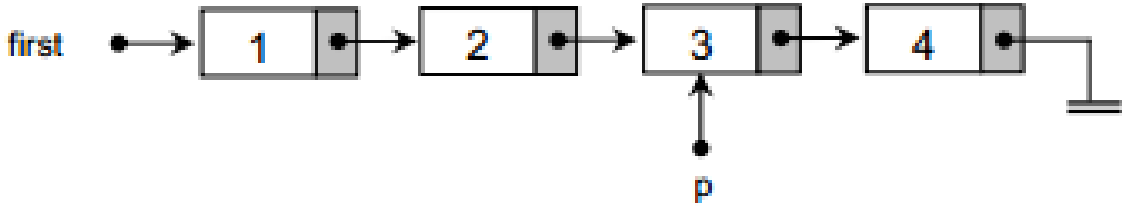
1. Telusuri simpul s/d first->next = NULL

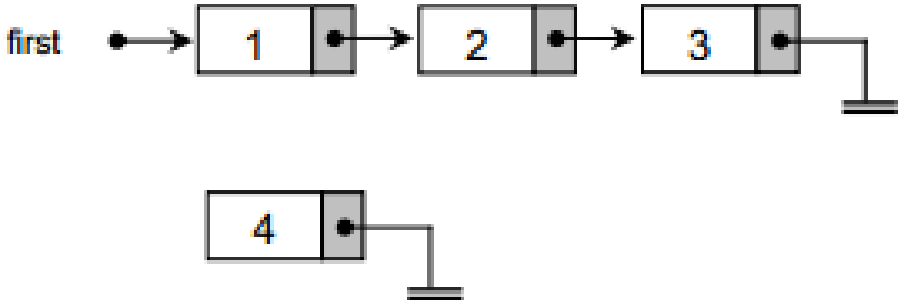
2. Arahkan pointer p ke first

3. Bebaskan pointer p->next (Simpul Terakhir)

4. Arahkan p->next ke NULL


first
1
2
3
4


first
1
2
3
4
p


first
1
2
3
4


first
1
2
3

2.3 Double Linked List

Elemen-elemen dihubungkan dengan dua pointer dalam satu elemen.

Struktur ini menyebabkan list melintas baik ke depan maupun ke belakang.

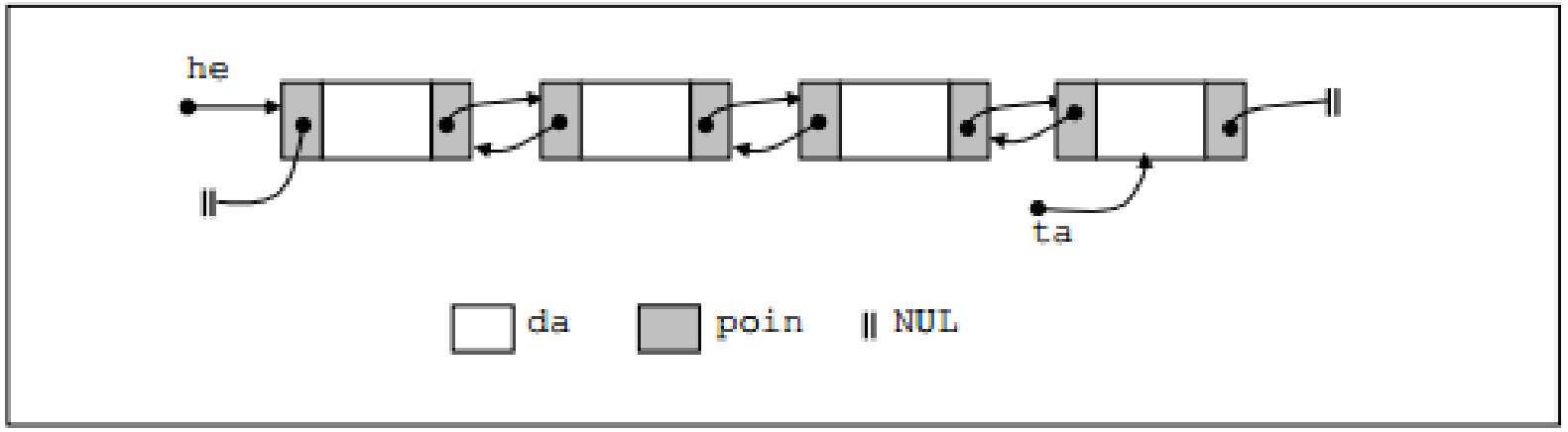
Masing-masing elemen pada double linked list terdiri dari tiga bagian, disamping data dan pointer next, masing-masing elemen dilengkapi dengan pointer prev yang menunjuk ke elemen sebelumnya.

Double linked list dibentuk dengan menyusun sejumlah elemen sehingga pointer next menunjuk ke elemen yang mengikutinya dan pointer prev menunjuk ke elemen yang mendahuluinya.

Untuk menunjukkan head dari double linked list, maka pointer prev dari elemen pertama menunjuk NULL.

2.3 Double Linked List

Untuk menunjukkan tail dari double linked list tersebut, maka pointer next dari elemen terakhir menunjuk NULL.



Untuk melintas kembali melalui double linked list, kita gunakan pointer prev dari elemen yang berurutan pada arah tail ke head.

Double linked list mempunyai fleksibilitas yang lebih tinggi daripada single linked list dalam perpindahan pada list.

Bentuk ini sangat berguna ketika akan meletakkan suatu elemen pada list dan dapat memilih dengan lebih bijaksana bagaimana memindahkannya.

2.4 Circular List

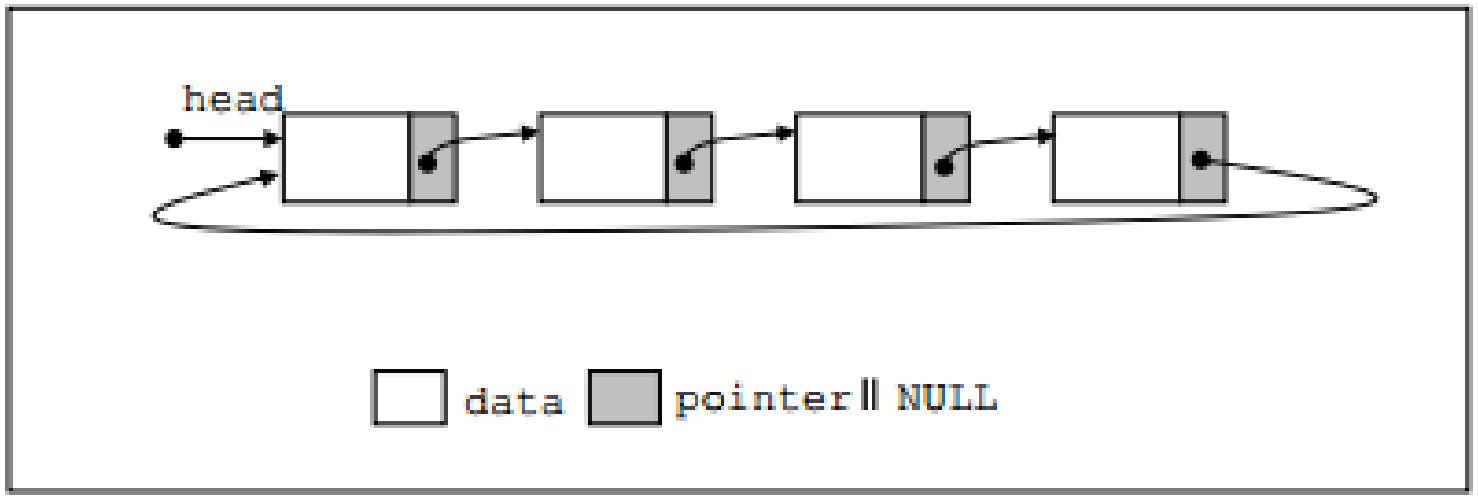
Circular list adalah bentuk lain dari linked list yang memberikan fleksibilitas dalam melewatkan elemen.

Circular list bisa berupa single linked list atau double linked list, tetapi tidak mempunyai tail.

Pada circular list, pointer next dari elemen terakhir menunjuk ke elemen pertama dan bukan menunjuk NULL.

Pada double linked circular list, pointer prev dari elemen pertama menunjuk ke elemen terakhir.

Hanya menangani link dari elemen terakhir kembali ke elemen pertama.


head
data
pointer
NULL

2.5 Kesimpulan

1. Linked list adalah sebuah struktur untuk menyimpan data yang bersifat dinamik

2. Beberapa operasi dapat diterapkan pada linked list seperti sisip(insert), hapus(delete)

3. Operasi-operasi yang ada pada linked list relatif lebih sulit jika dibandingkan dengan operasi-operasi pada struktur yang statis

Terima Kasih

Pertanyaan?