Pointer

DEFINISI

- **Pointer** adalah suatu variabel penunjuk yang menunjuk alamat dari sebuah lokasi memori tertentu.

- Pointer tidak berisi nilai data, tapi berisi sebuah alamat memori atau null jika tidak berisi data.

  Dengan kata lain, pointer berisi alamat dari variabel yang mempunyai nilai tertentu.

- Bentuk umum dari pernyataan variabel pointer :

  tipe_data *nama_variabel;

  dimana :

  - tipe_data adalah tipe dasar pointer
  - nama_variabel adalah nama variabel pointer
  - * adalah operator memori yang fungsinya untuk mengembalikan nilai variabel pada alamatnya yang ditentukan oleh operand

- Contoh :

  float *ABC;

  → Statement diatas mendeklarasikan sebuah variabel bernama ABC sebagai pointer, dimana alamat memori yang ditunjuk oleh ABC dimaksudkan untuk berisi data bertipe float

- Perbedaan pointer dengan variabel biasa :

Pointer	Variabel biasa
Berisi alamat memori dari suatu variabel tertentu	Berisi data/nilai
Membutuhkan operator khusus "&" yang menunjuk alamat dari suatu variabel tertentu dan operator "*" yang membaca nilai dari alamat variabel yang ditunjuk oleh pointer tersebut	Operasi yang bisa dilakukan adalah : • operasi aritmatik → +, -, *, /, % • operator rasional → <, >, <=, >=, ==, != • operator logika → &&,||,! • operator assignment → +=, -=, *=, /=, %=
Pengalokasiannya bersifat dinamis	Pengalokasiannya bersifat statis
Deklarasi : tipe_data *nama_var;	Deklarasi : tipe_data nama_var;

- Operator pada pointer ada dua macam, yaitu :

1. **Operator Deference (&)**

   → Untuk mengetahui alamat memori tempat penyimpanan data.

   Contoh :

```
int *X;
int Y=100;

X=&Y;
cout<<X;
```

   Maka hasilnya adalah : 0x0012ff50

2. **Operator Reference (*)**

→ Untuk mengakses nilai/data pada sebuah alamat ` memori yang ditunjuk oleh variabel pointer.

Contoh :

```
int *X;
int Y=100,Z;

X=&Y;
Z=*X;
cout<<Z;
```

Maka hasilnya adalah : 100

- Contoh implementasi pointer :

```
#include <iostream.h>
#include <conio.h>

main()
{
    int *A,*B;
    int nilai1=100,nilai2=200;

    A=&nilai1;
    B=&nilai2;
    cout<<"Data "<<*A<<" disimpan pada alamat :  "<<&A;
    cout<<"Data "<<*B<<" disimpan pada alamat :  "<<&B;
    getch();
}
```

Linked List
(Senarai Berantai)

Pendahuluan

- Dalam pemrograman, untuk proses penyimpanan banyak data maka menggunakan variabel array.

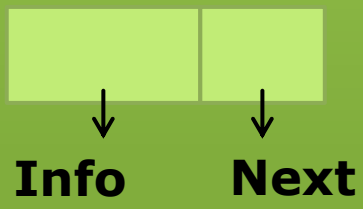
  Namun variabel array ini bersifat statis (ukuran dan urutannya sudah pasti). Selain itu, ruang memori yang dipakai oleh variabel array tidak dapat dihapus bila array tersebut sudah tidak digunakan lagi pada saat program dijalankan.

- Sehingga dibutuhkan variabel yang bersifat dinamis / *dynamic variable,* dimana variabel akan dialokasikan hanya pada saat dibutuhkan dan sesudah tidak dibutuhkan dapat direlokasikan kembali, yang disebut dengan ***LINKED LIST***

Linked List

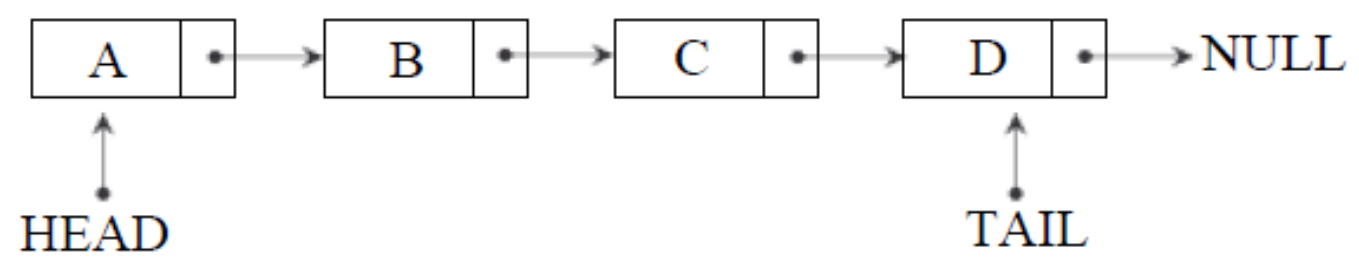
- ***Linked List*** adalah sejumlah simpul (node) yang dihubungkan secara linier dengan bantuan pointer. Sebuah simpul dapat berupa struktur.

- Sebuah simpul harus paling sedikit terdiri dari dua bagian/field, yaitu :
  1. Medan informasi (info), berisi informasi atau data yang akan disimpan/diolah.
  2. Medan penyambung/pointer (next), berisi alamat memori dari simpul/node berikutnya.



Single Linked List

- Single Linked List adalah sebuah list dimana satu simpul/node hanya memiliki sebuah variabel pointer yang digunakan untuk menunjuk pada simpul/node berikutnya.
- Ilustrasi Single Linked List :



Single Linked List diatas terdiri dari 4 simpul. Simpul A disebut **kepala/head** dan simpul D disebut **ekor/tail**. Tanda panah merupakan pengait/pointer yang menghubungkan satu simpul dengan simpul yang lainnya. Pointer yang dimiliki oleh simpul D, karena tidak mengait ke simpul lain maka menuju ke NULL.

Single Linked List

- Bentuk umum deklarasi *single linked list* :

```
struct tipe_data pointer
{
    tipe_data nama_medan_info;
    tipe_data_pointer *nama_medan_penyambung;
};
tipe_data_pointer *nama_variabel_pointer;
```

Keterangan :

- tipe_data_pointer = tipe data buatan yang berupa simpul/node
- nama_medan_info = nama medan informasi
- nama_medan_penyambung =nama medan penyambung/pointer
- nama_variabel_pointer = nama variabel yang bertipe pointer

Single Linked List

- Contoh :

```
struct simpul
{
    char data;
    simpul *next;
};
simpul *X;
```

Dari contoh diatas, berarti telah dibuat sebuah tipe data buatan berupa struktur yang diberi nama “simpul”. Dimana field yang terbentuk ada 2, yaitu data →untuk menyatakan meda informasi dan next → untuk menyatakan medan penyambung (harus bertipe pointer).

Single Linked List

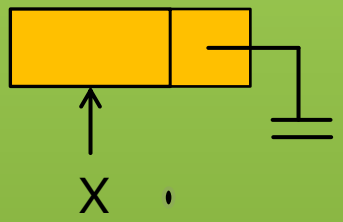
- Untuk membuat simpul/mengalokasikan sebuah simpul dalam memori, menggunakan statement :

```
new nama_simpul;
```

- Contoh :

```
X=new simpul;
```

Dengan perintah tersebut maka akan terbentuk sebuah simpul baru yang ditunjuk oleh pointer X.



- Pada saat pertama kali, *Linked list* selalu dalam keadaan kosong. Hal ini dikarenakan belum ada simpul yang terbentuk.

Operasi pada Single Linked List

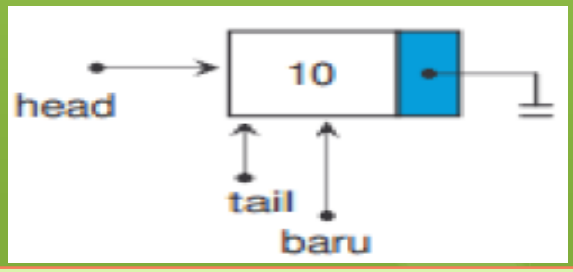
1. Membentuk Simpul Awal

→ Membuat sebuah simpul dalam linked list :

Algoritma :

```
baru=new simpul;
baru->data=10;
baru->next=NULL;
head=baru;
tail=baru;
```

Dari algoritma tersebut maka terbentuk :



Operasi pada Single Linked List

2. Menyisipkan Simpul

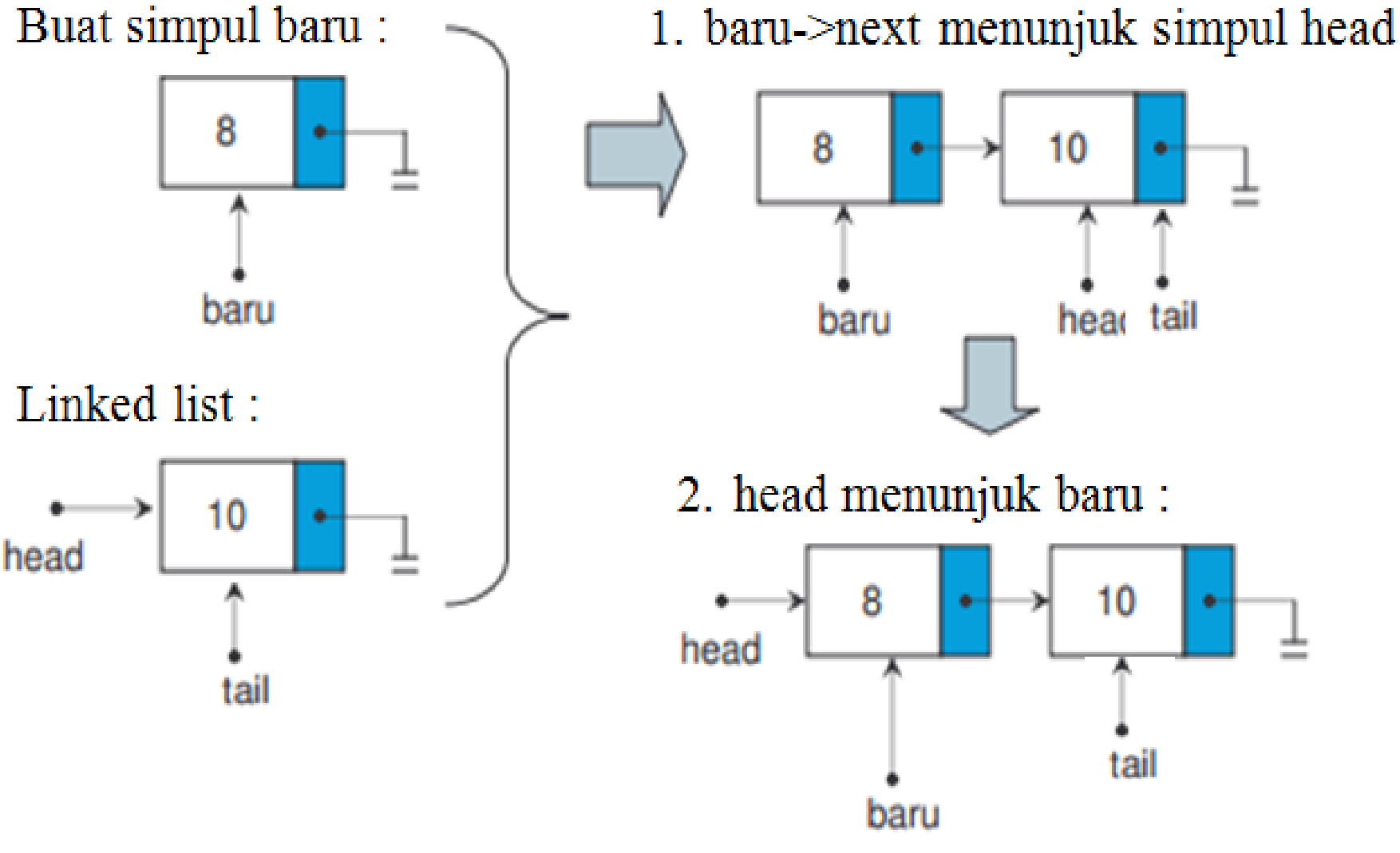
→ Menyisipkan atau menambahkan sebuah simpul ke dalam linked list :

a. Menyisipkan di depan linked list

Algoritma :

```
baru=new simpul;
baru->data=elemen;
baru->next=NULL;
if (head==NULL)
    tail=baru;
else
    baru->next=head;
head=baru;
```

Ilustrasi menyisipkan di depan linked list :

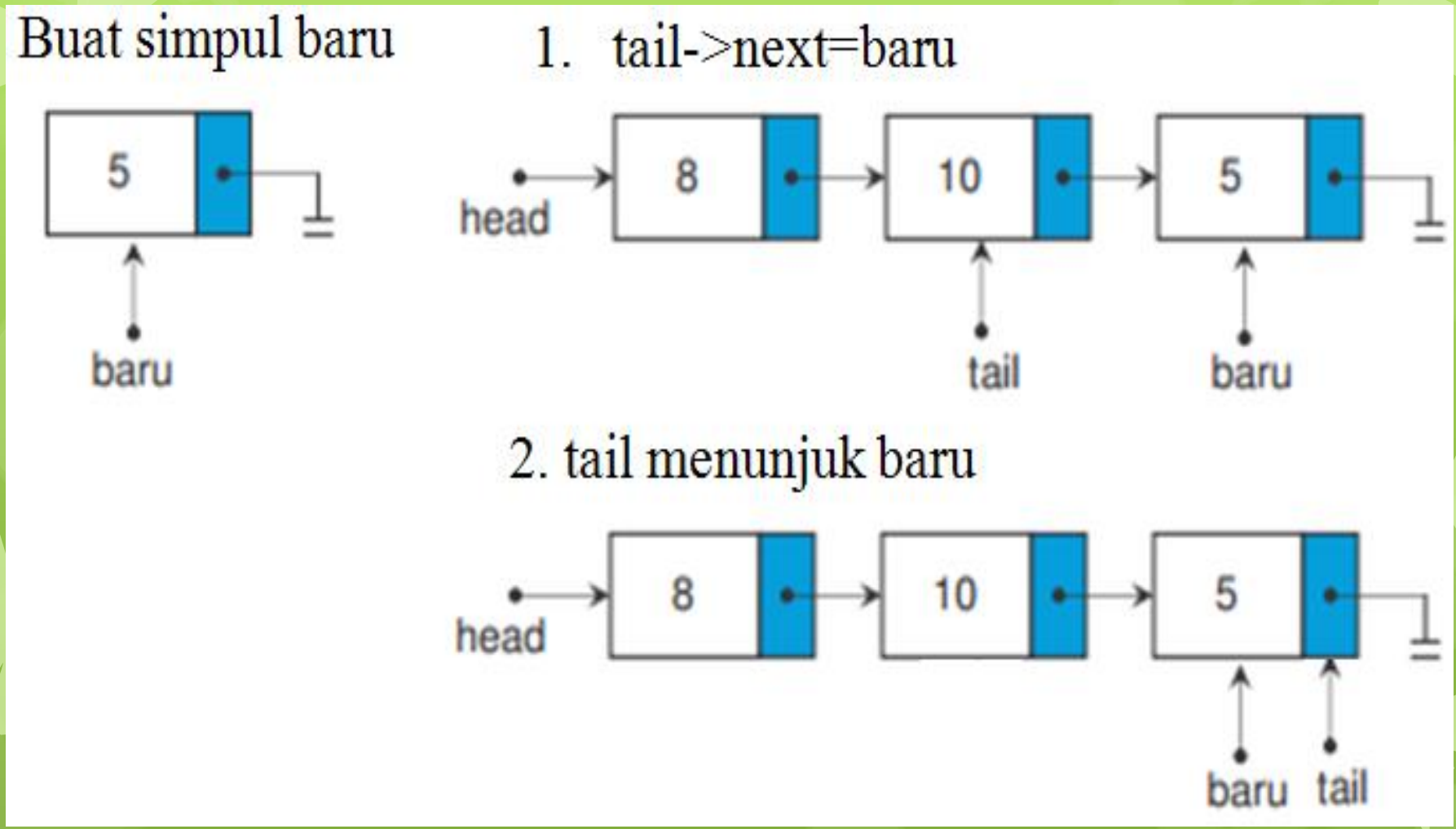


b. Menyisipkan di belakang linked list

Algoritma :

```
baru=new simpul;
baru->data=elemen;
baru->next=NULL;
if (head==NULL)
    head=baru;
else
    tail->next=baru;
tail=baru;
```

Ilustrasi menyisipkan di depan linked list :



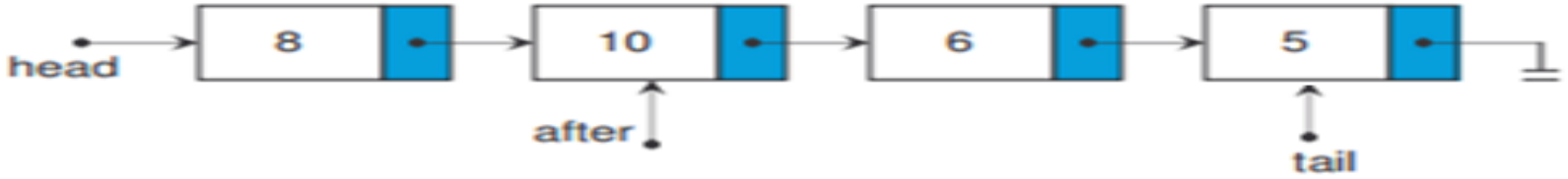
c. Menyisipkan di tengah linked list

Algoritma :

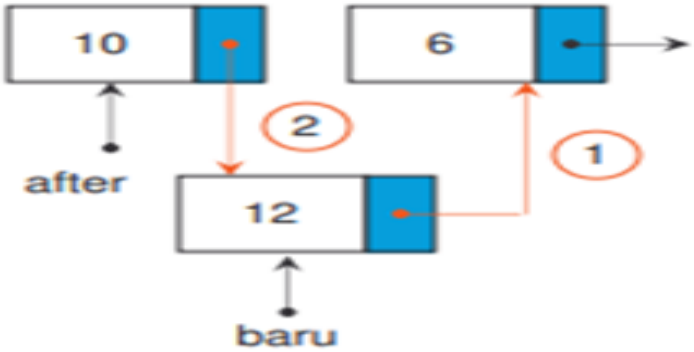
```
baru=new simpul;
baru->data=elemen;
baru->next=NULL;
posisi=elemen;
if (head==NULL)
{     head=baru; tail=baru;    }
else
{     bantu=head;
      while (bantu->data != posisi)
            bantu=bantu->next;
      baru->next=bantu->next;
      bantu->next=baru     }
```

Ilustrasi menyisipkan di tengah linked list :

1. after=head
2. Arahkan after sampai pada simpul dengan data=10



3. baru->next=after->next
4. after->next=baru



Operasi pada Single Linked List

3. Menghapus Simpul

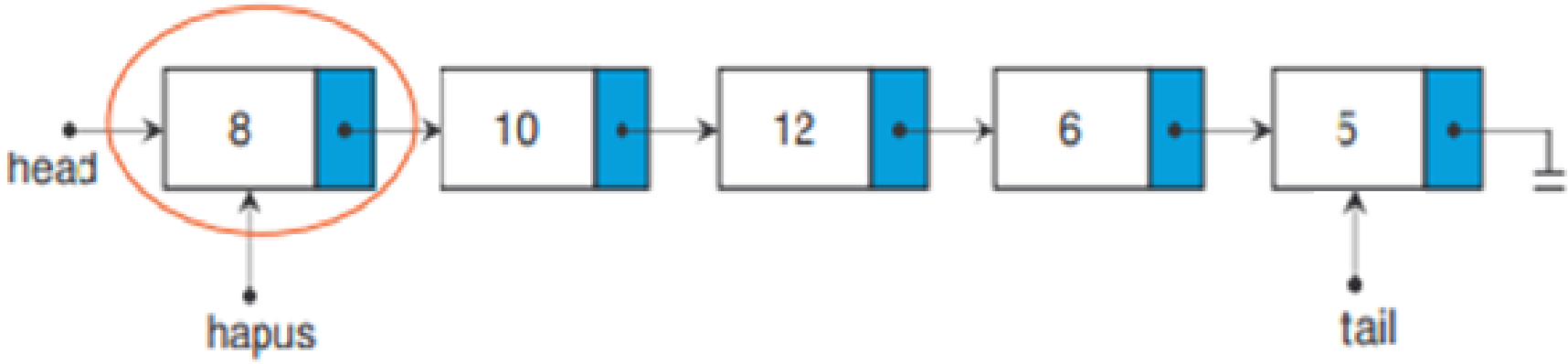
→ Menghapus simpul dari pada Linked list. Proses penghapusan simpul hanya dapat dilakukan jika Linked List tidak kosong :

a. Menghapus di depan linked list

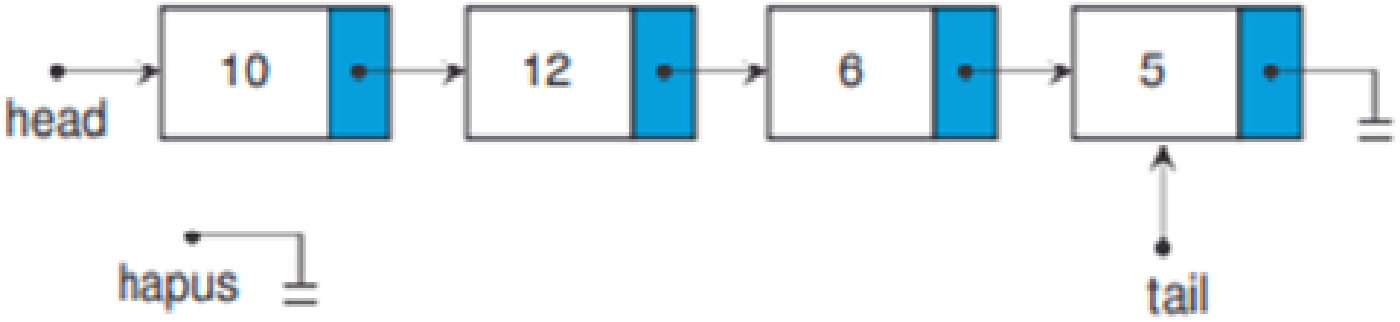
Algoritma :

```
hapus=head;

head=hapus->next;

hapus=NULL
```

Ilustrasi menghapus di depan linked list :



2. head=hapus->next , hapus=NULL;



Operasi pada Single Linked List

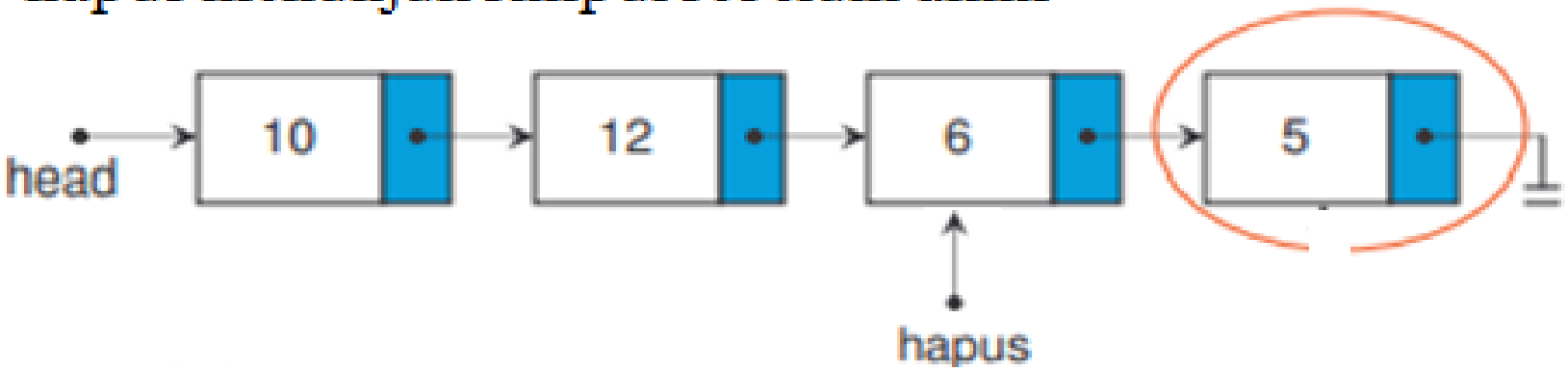
a. Menghapus di belakang linked list

Algoritma :

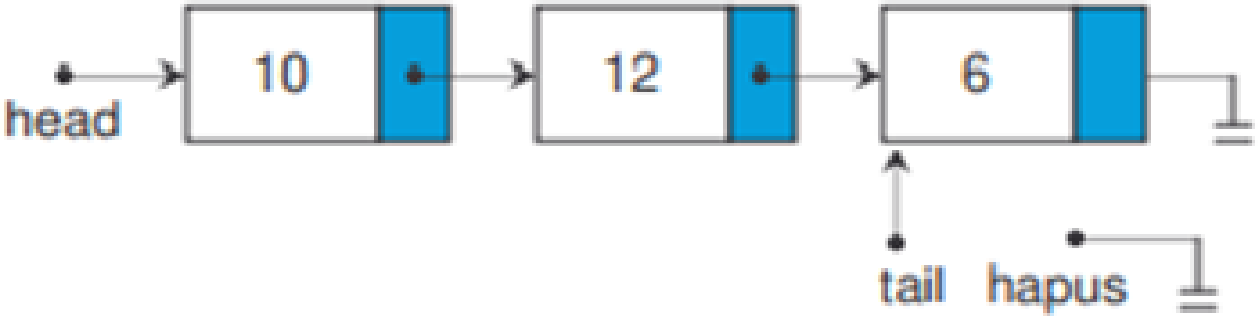
```
hapus=head;
while (hapus->next->next != NULL)
    hapus=hapus->next;
tail=hapus;
hapus->next=NULL;
```

Ilustrasi menghapus di belakang linked list :

1. hapus menunjuk simpul sebelum akhir



2. tail=hapus , hapus->next=NULL;



Operasi pada Single Linked List

a. Menghapus di tengah linked list

Algoritma :

```
posisi=elemen;
hapus=head;
while (hapus->data != posisi)
{
        bantu=hapus; hapus=hapus->next;
}
bantu->next=hapus->next;
hapus=NULL;
```

Operasi pada Single Linked List

4. Mencetak Simpul

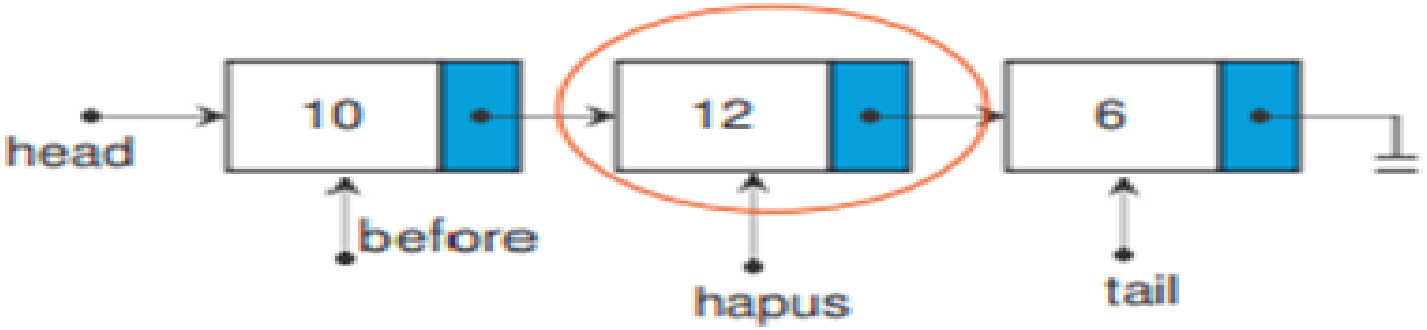
→ Mencetak simpul yang ada dalam linked list

Algoritma :

```
bantu=head;
while (bantu != NULL)
{
    cout<<bantu->data;
    bantu=bantu->next;
}
```

Ilustrasi menghapus di tengah linked list :

1. Bila letak=12, arahkan hapus sampai pada simpul dengan data = 12



2. before->next=hapus->next , hapus=NULL