Pertemuan 4

SINGLE LINKED LIST
(Non Circular)

KONSEP POINTER DAN LINKED LIST

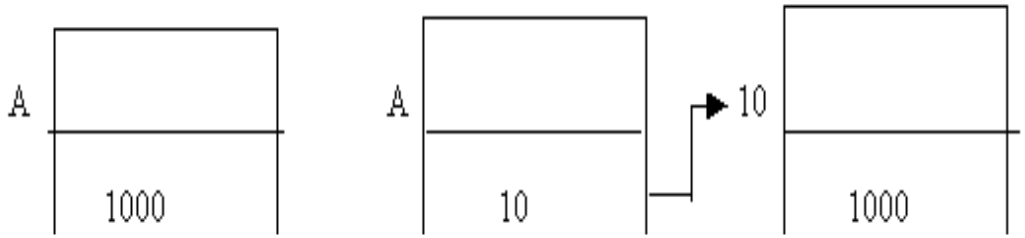
Untuk mengolah data yang banyaknya tidak bisa ditentukan sebelumnya, maka disediakan satu fasilitas yang memungkinan untuk menggunakan suatu perubah yang disebut dengan perubah dinamis (Dinamic variable)

Perubah Dinamis (Dinamic variable)
Suatu perubah yang akan dialokasikan hanya pada saat diperlukan, yaitu setelah program dieksekusi.

Perbedaan Perubah Statis & Dinamis

Pada perubah statis, isi Memory pada lokasi tertentu (nilai perubah) adalah data sesungguhnya yang akan diolah. Pada perubah dinamis, nilai perubah adalah alamat lokasi lain yang menyimpan data sesungguhnya. Dengan demikian data yang sesungguhnya dapat dimasukkan secara langsung.

Dalam hal cara pemasukkan data dapat diilustrasikan seperti dibawah ini.



DEKLARASI POINTER

Pointer digunakan sebagai penunjuk ke suatu alamat memori

Dalam pemrograman C++, Type Data Pointer

dideklarasikan dengan bentuk umum :

```
Type Data *Nama Variabel;
```

Type Data dapat berupa sembarang type data, misalnya char, int atau float. Sedangkan Nama veriabel merupakan nama variabel pointer

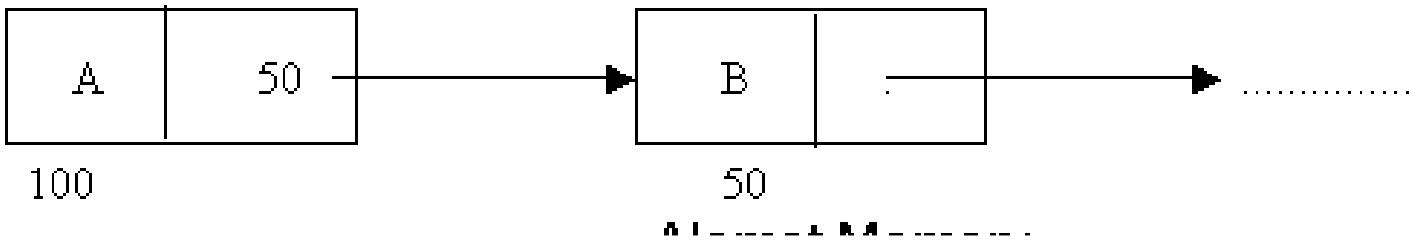
Contoh penggunaan pointer dalam program C++:

```
Void main()
{
  int x,y,*z;
  x = 75;       //nilai x = 75
  y = x;        //nilai y diambil dari nilai x
  z = &x;      //nilai z menunjuk kealamat pointer dari nilai
x
  getch();
}
```

LINKED LIST (LINKED LIST)

Salah satu Struktur Data Dinamis yang paling sederhana adalah Linked List atau Struktur Berkait atau Senarai Berantai, *yaitu suatu kumpulan komponen yang disusun secara berurutan dengan bantuan Pointer*.

Linked List (Senarai Berantai) disebut juga dengan Senarai Satu Arah (One-Way List). Masing-masing komponen dinamakan dengan Simpul (Node).



Perbedaan Karakteristik Array dan Linked List

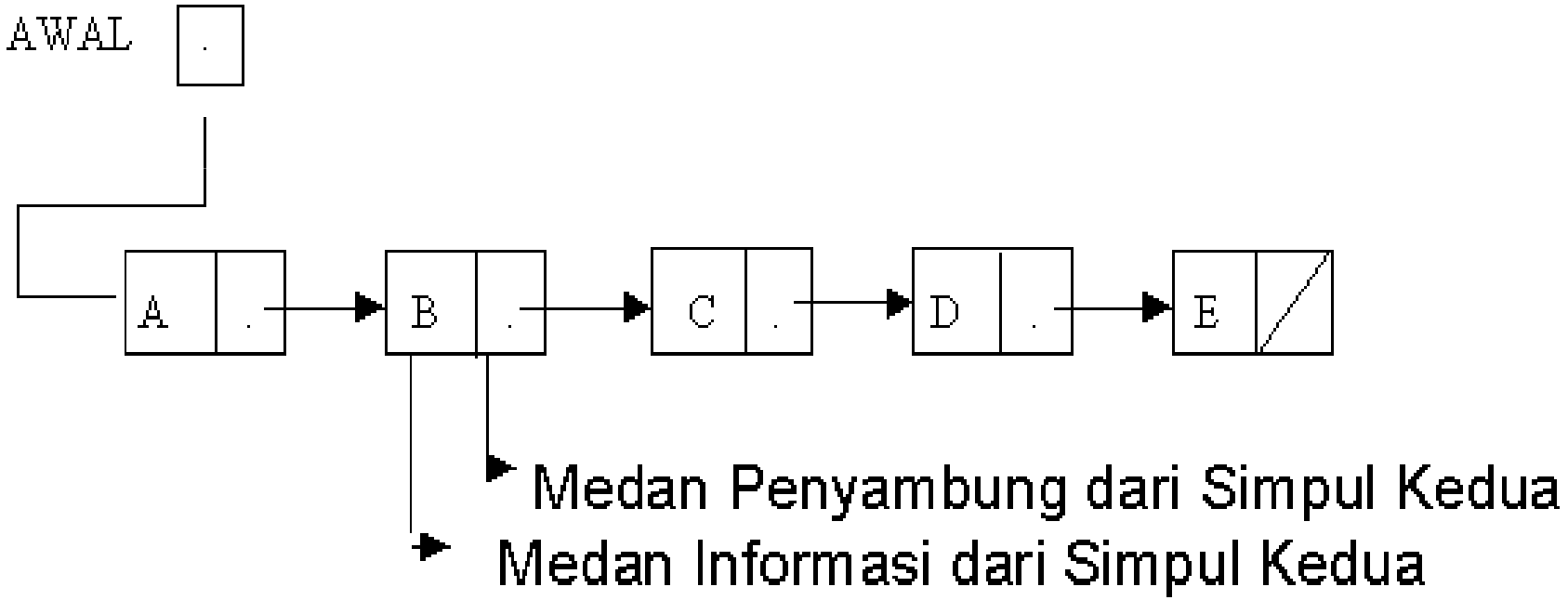
ARRAY	LINKED LIST
Statis	Dinamis
Penambahan / penghapusan data terbatas	Penambahan / penghapusan data tidak terbatas
Random access	Sequential access
Penghapusan array tidak mungkin	Penghapusan linked list mudah

Setiap simpul dalam suatu Linked List terbagi menjadi dua bagian,yaitu :

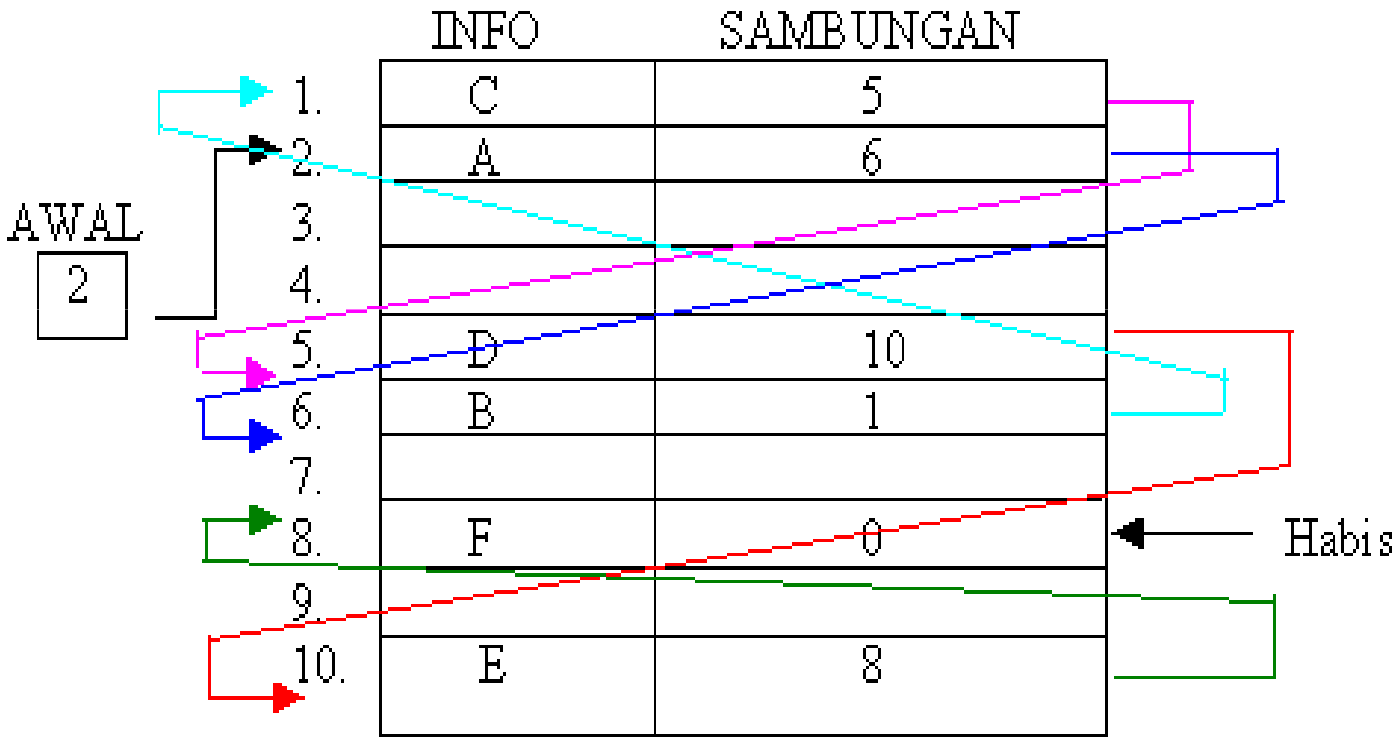
1. Medan Informasi
   Berisi informasi yang akan disimpan dan diolah.

2. Medan Penyambung (Link Field)
   Berisi alamat berikutnya. Bernilai 0, Jika Link tersebut tidak menunjuk ke Data (Simpul) lainnya. Penunjuk ini disebut Penunjuk Nol.

Linked List juga mengandung sebuah variabel Penunjuk List, yang biasanya diberi nama START(AWAL), yang berisi alamat dari simpul pertama dalam List



PENYAJIAN LINKED LIST DALAM MEMORY



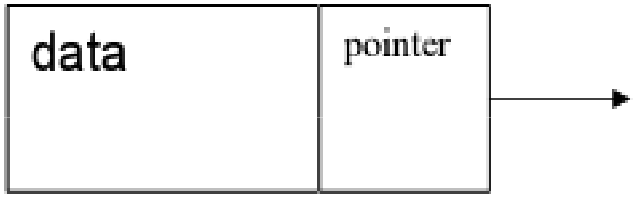
Keterangan :

AWAL	=	2	,	Maka	INFO[2]	=	'A'
SAMBUNGAN[2]	=	6	,	Maka	INFO[6]	=	'B'
SAMBUNGAN[6]	=	1	,	Maka	INFO[1]	=	'C'
SAMBUNGAN[1]	=	5	,	Maka	INFO[5]	=	'D'
SAMBUNGAN[5]	=	10	,	Maka	INFO[10]	=	'E'
SAMBUNGAN[10]	=	8	,	Maka	INFO[8]	=	'F'
SAMBUNGAN[8]	=	0	,	Maka	Akhir Linked List		

Dari contoh diatas diperoleh untai **'ABCDEF'**

Bentuk Node
Single Linked List non Circular

- Single : field pointer-nya hanya satu dan satu arah,pada akhir node pointernya menunjuk NULL

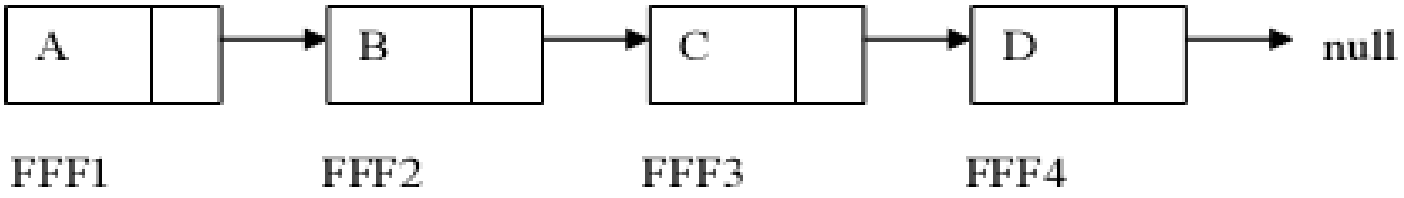


Menempati alamat memori tertentu

- Linked List : node-node tersebut saling terhubung satu sama lain.

- Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.

- Node terakhir akan menunjuk ke NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.



Pembuatan
Single Linked List non Circular

Deklarasi Node :

```
typedef struct TNode{
    int data;
    TNode *next;
};
```

Keterangan:

- Pembuatan struct bernama TNode yang berisi 2 field, yaitu field data bertipe integer dan field **next** yang bertipe pointer dari TNode

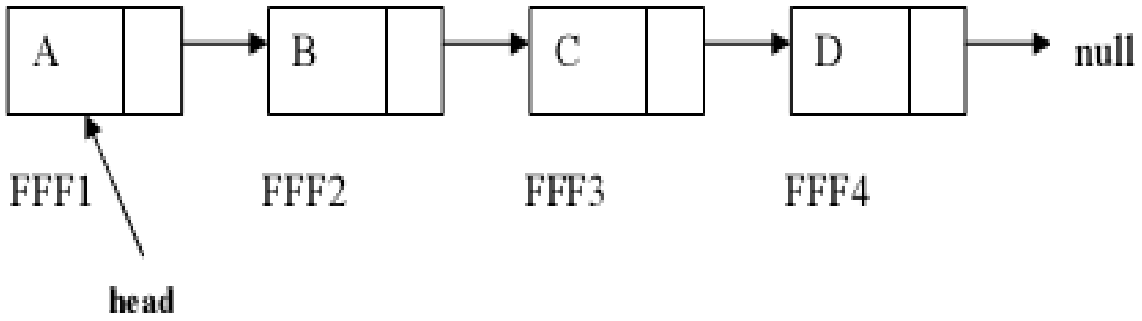
- Setelah pembuatan struct, buat variabel head yang bertipe pointer dari TNode yang berguna sebagai kepala linked list.

- Digunakan perintah **new** untuk mempersiapkan sebuah node baru berserta alokasi memorinya, kemudian node tersebut diisi data dan pointer nextnya ditunjuk ke NULL.

```
TNode *baru;
baru = new TNode;
baru->data = databaru;
baru->next = NULL;
```

Single Linked List non Circular Menggunakan Head

- Dibutuhkan satu buah variabel pointer : **head** yang akan selalu menunjuk pada **node pertama**



Deklarasi Pointer Penunjuk Head Single Linked List

- Manipulasi linked list tidak dapat dilakukan langsung ke node yang dituju, melainkan harus menggunakan suatu pointer penunjuk ke node pertama (Head) dalam linked list

- Deklarasinya sebagai berikut:

TNode *head;

Fungsi Inisialisasi Single Linked List

```
void init()
{
    head = NULL;
}
```

Function untuk mengetahui kondisi Single Linked List

- Jika pointer head tidak menunjuk pada suatu node maka kosong

```
int isEmpty()
{
    if (head == NULL) return 1;
    else return 0;
}
```

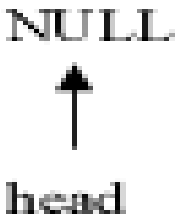
Menambah Node di Depan

- Penambahan node baru akan dikaitan di node **paling depan,** namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan dengan cara: node head ditunjukkan ke node baru tersebut.

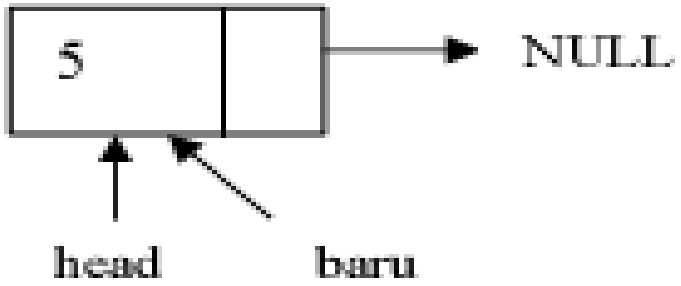
- Prinsipnya adalah mengkaitkan node baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan.

```
void insertDepan(int databaru)
{
  TNode *baru;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1)
    {
        head=baru;
        head->next = NULL;
    }
  else
   {
        baru->next = head;
        head = baru;
   }
  printf(”Data masuk\n”);
}
```

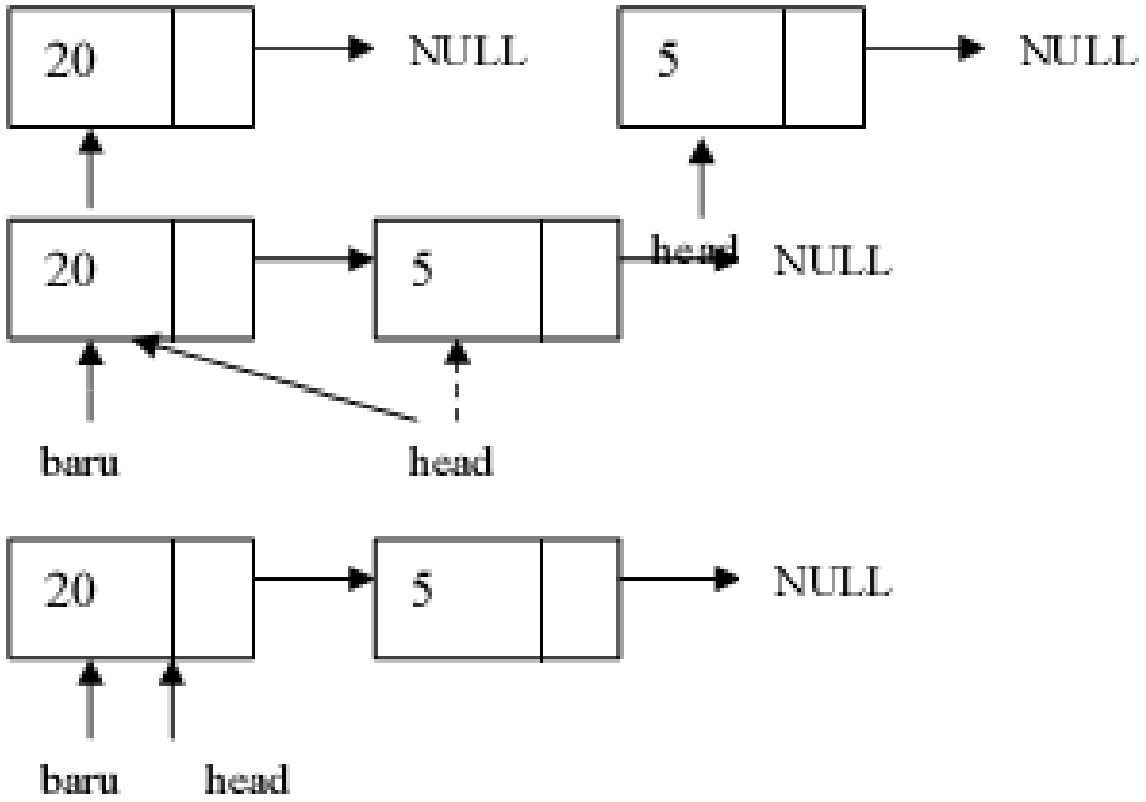
1. List masih kosong (`head=NULL`)



2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20 (penambahan di depan)



Menambah Node di Belakang

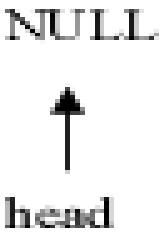
- Penambahan data dilakukan **di belakang**, namun pada saat pertama kali, node langsung ditunjuk oleh head.

- Penambahan di belakang membutuhkan pointer bantu untuk mengetahui node terbelakang. Kemudian, dikaitkan dengan node baru.

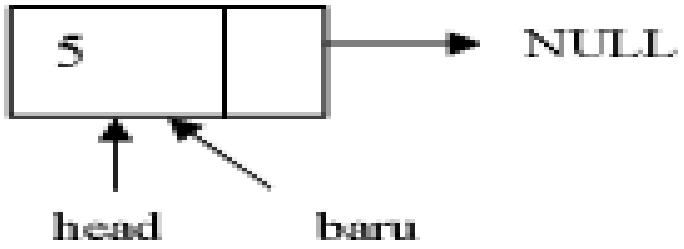
- Untuk mengetahui data terbelakang perlu digunakan perulangan.

```
void insertBelakang (int databaru)
{
  TNode *baru,*bantu;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1) {
    head=baru;
    head->next = NULL;
  }
  else {
    bantu=head;
    while(bantu->next!=NULL){
       bantu=bantu->next;
    }
    bantu->next = baru;
  }
  printf("Data masuk\n“);
}
```

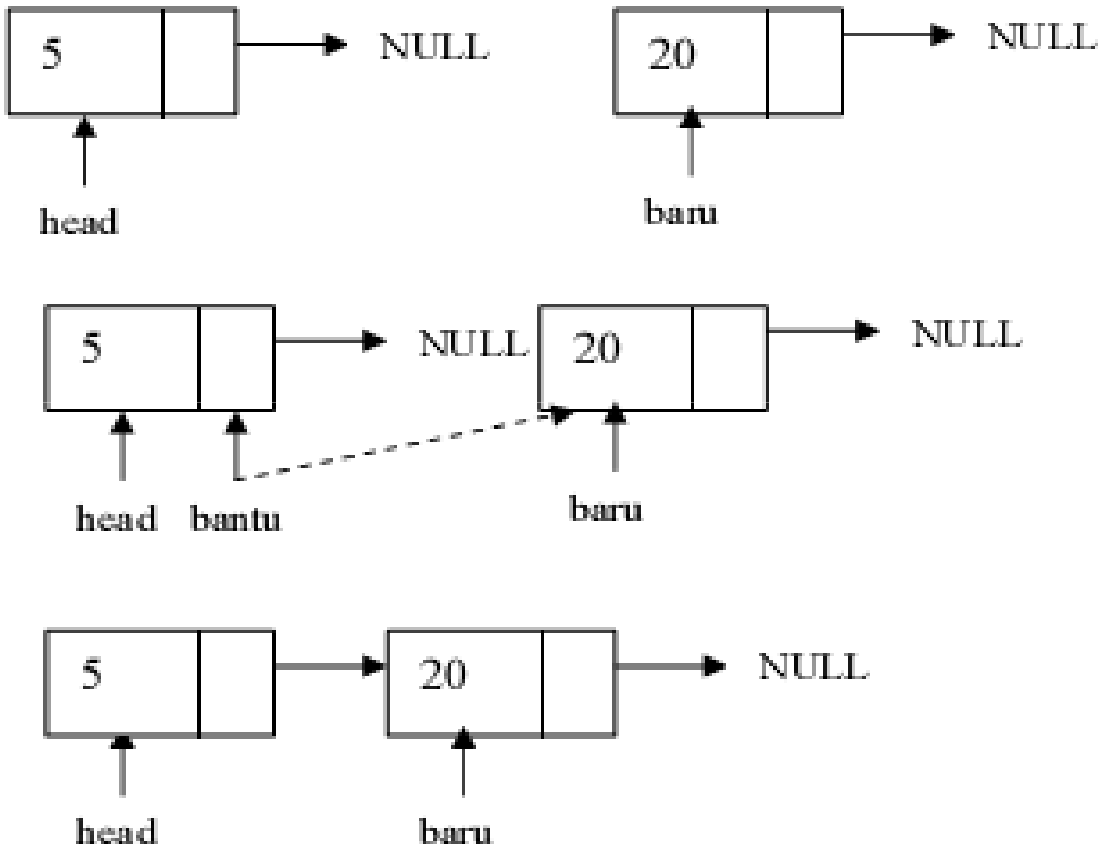
1. List masih kosong (`head=NULL`)



2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20 (penambahan di belakang)



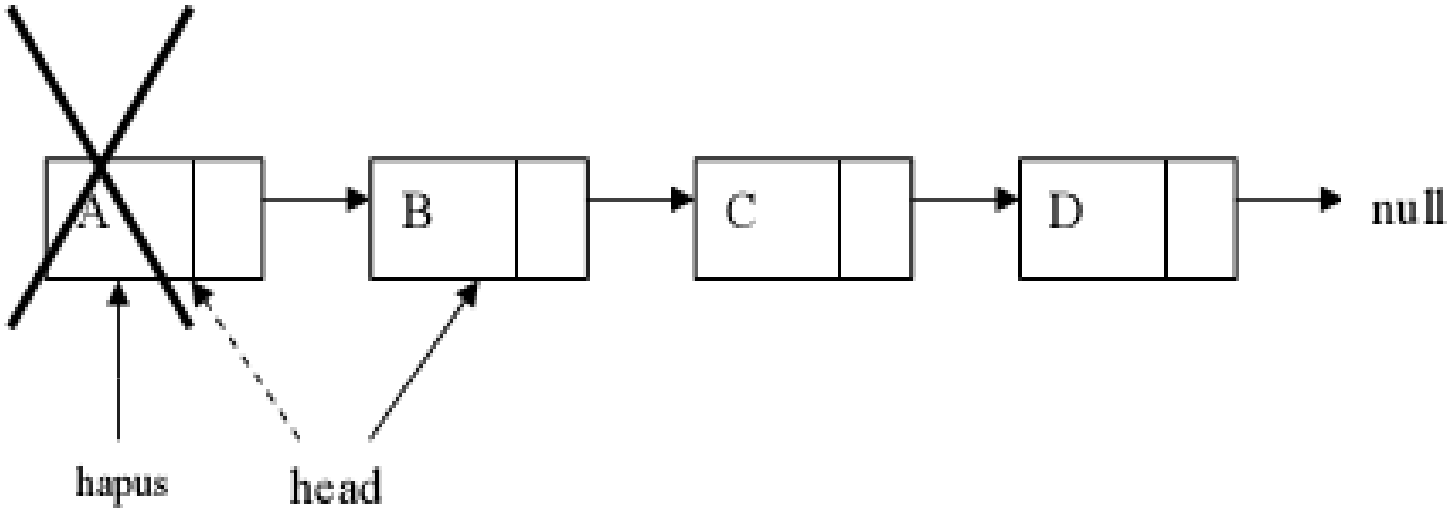
Menghapus Node di Depan

- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penggunakan suatu pointer lain (hapus) yang digunakan untuk menunjuk node yang akan dihapus, barulah kemudian menghapus pointer hapus dengan menggunakan perintah delete.

- Sebelum data terdepan dihapus, terlebih dahulu head harus menunjuk ke node berikutnya agar list tidak putus, sehingga node setelah head lama akan menjadi head baru

- Jika head masih NULL maka berarti data masih kosong!

```
void hapusDepan ()
{
    TNode *hapus;
    int d;
    if (isEmpty()==0){
      if(head->next != NULL){
            hapus = head;
            d = hapus->data;
            head = head->next;
            delete hapus;
     } else {
            d = head->data;
            head = NULL;
     }
     printf("%d terhapus\n",d);
    } else cout<<"Masih kosong\n";
}
```

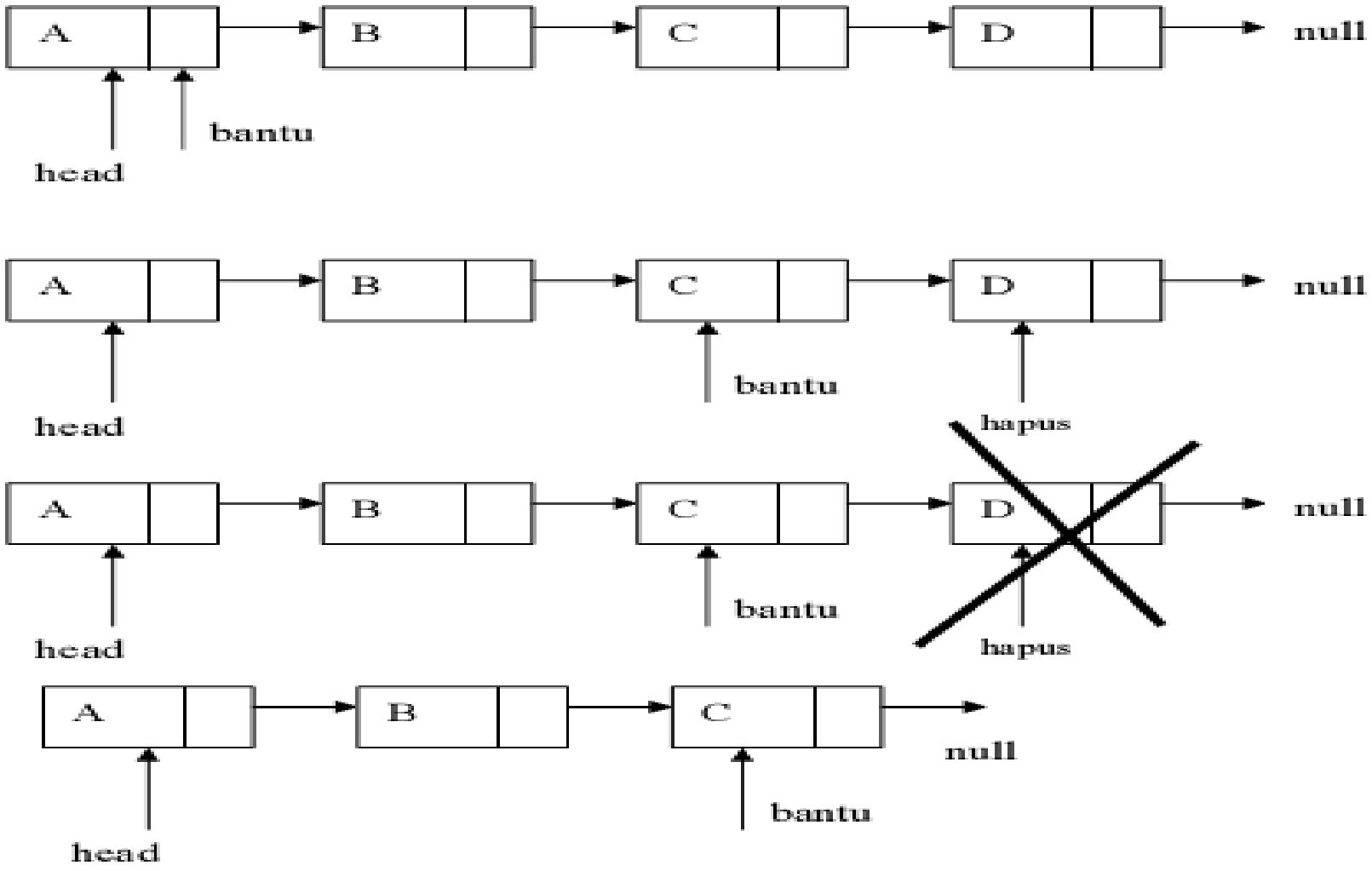


Menghapus Node di Belakang

- Membutuhkan pointer bantu dan hapus. Pointer hapus digunakan untuk menunjuk node yang akan dihapus, pointer bantu untuk menunjuk node sebelum node yang dihapus yang akan menjadi node terakhir.

- Pointer bantu digunakan untuk menunjuk ke nilai NULL. Pointer bantu selalu bergerak sampai sebelum node yang akan dihapus, kemudian pointer hapus diletakkan setelah pointer bantu. Selanjutnya pointer hapus akan dihapus, pointer bantu akan menunjuk ke NULL.

```
void hapusBelakang(){
    TNode *hapus,*bantu;
    int d;
    if (isEmpty()==0){
     if(head->next != NULL){
            bantu = head;
            while(bantu->next->next!=NULL){
              bantu = bantu->next;
            }
            hapus = bantu->next;
            d = hapus->data;
                    bantu->next = NULL;
            delete hapus;
     } else {
            d = head->data;
            head = NULL;
     }
     printf("%d terhapus\n",d);
    } else printf("Masih kosong\n");
}
```



Function untuk menghapus semua elemen Linked List

```
void clear()
{
TNode *bantu,*hapus;
bantu = head;
    while(bantu!=NULL)
    {
    hapus = bantu;
    bantu = bantu->next;
    delete hapus;
    }
head = NULL;
}
```

Menampilkan / Membaca Isi Linked List

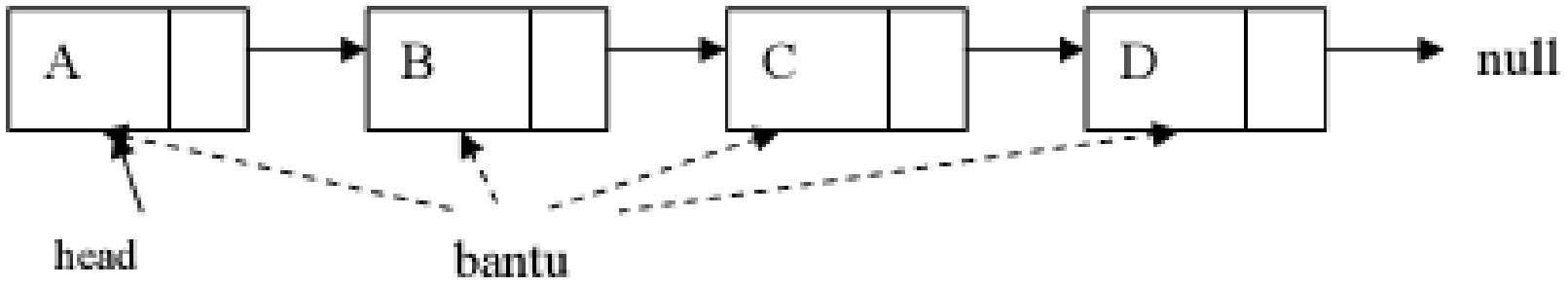
- Linked list ditelusuri satu-persatu dari awal sampai akhir node. Penelusuran dilakukan dengan menggunakan pointer bantu, karena pointer head yang menjadi tanda awal list tidak boleh berubah/berganti posisi.

- Penelusuran dilakukan terus sampai ditemukan node terakhir yang menunjuk ke nilai NULL.

  Jika tidak NULL, maka node bantu akan berpindah ke node selanjutnya dan membaca isi datanya dengan menggunakan field next sehingga dapat saling berkait.

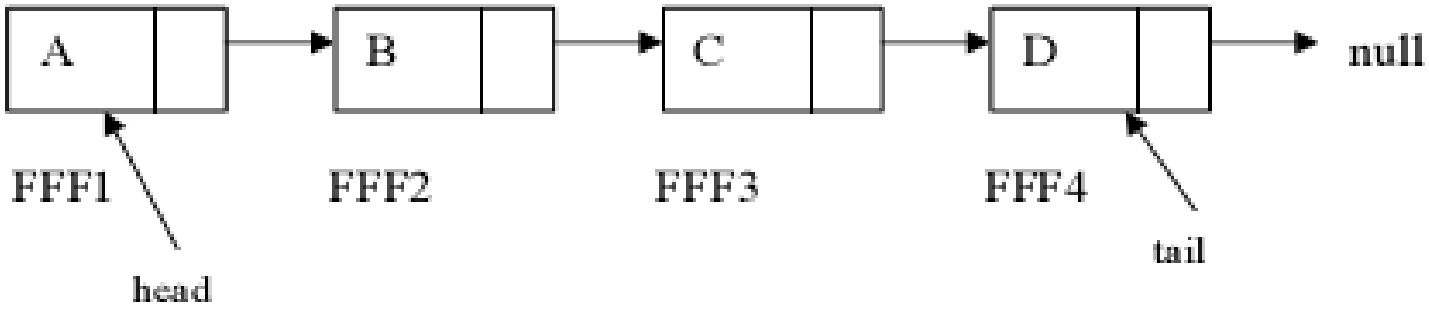
- Jika head masih NULL berarti data masih kosong!

```
void tampil(){
        TNode *bantu;
        bantu = head;
        if(isEmpty()==0){
                while(bantu!=NULL){
                        cout<<bantu->data<<" ";
                        bantu=bantu->next;
                }
                printf("\n");
        } else printf("Masih kosong\n");
}
```



Single Linked List non Circular Menggunakan Head dan Tail

- Dibutuhkan dua variabel pointer : **head** dan **tail**
- Head selalu menunjuk pada node **pertama**, sedangkan tail selalu menunjuk pada node **terakhir**.
- Kelebihan dari Single Linked List dengan Head & Tail adalah pada penambahan data di belakang, hanya dibutuhkan tail yang mengikat node baru saja tanpa harus menggunakan perulangan pointer bantu.



Inisialisasi Linked List

```
TNode *head, *tail;
```

Fungsi Inisialisasi Linked List

```
void init(){
   head = NULL;
   tail = NULL;
}
```

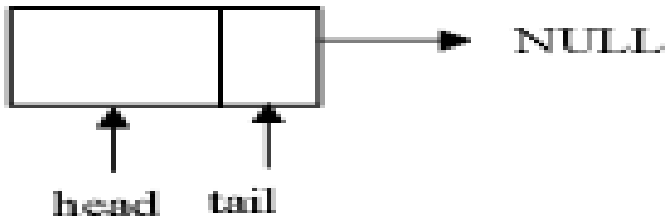
Function untuk mengetahui kondisi LinkedList kosong / tidak

```
int isEmpty(){
   if(tail == NULL) return 1;
   else return 0;
}
```

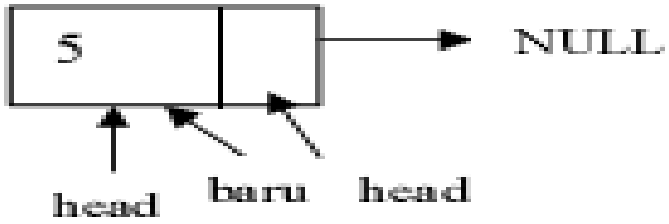
Menambah Node di Depan Dengan Head dan Tail

```
void insertDepan(int databaru){
  TNode *baru;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1){
      head=tail=baru;
      tail->next=NULL;
    }
  else {
    baru->next = head;
    head = baru;
  }
  printf(”Data masuk\n”);
}
```

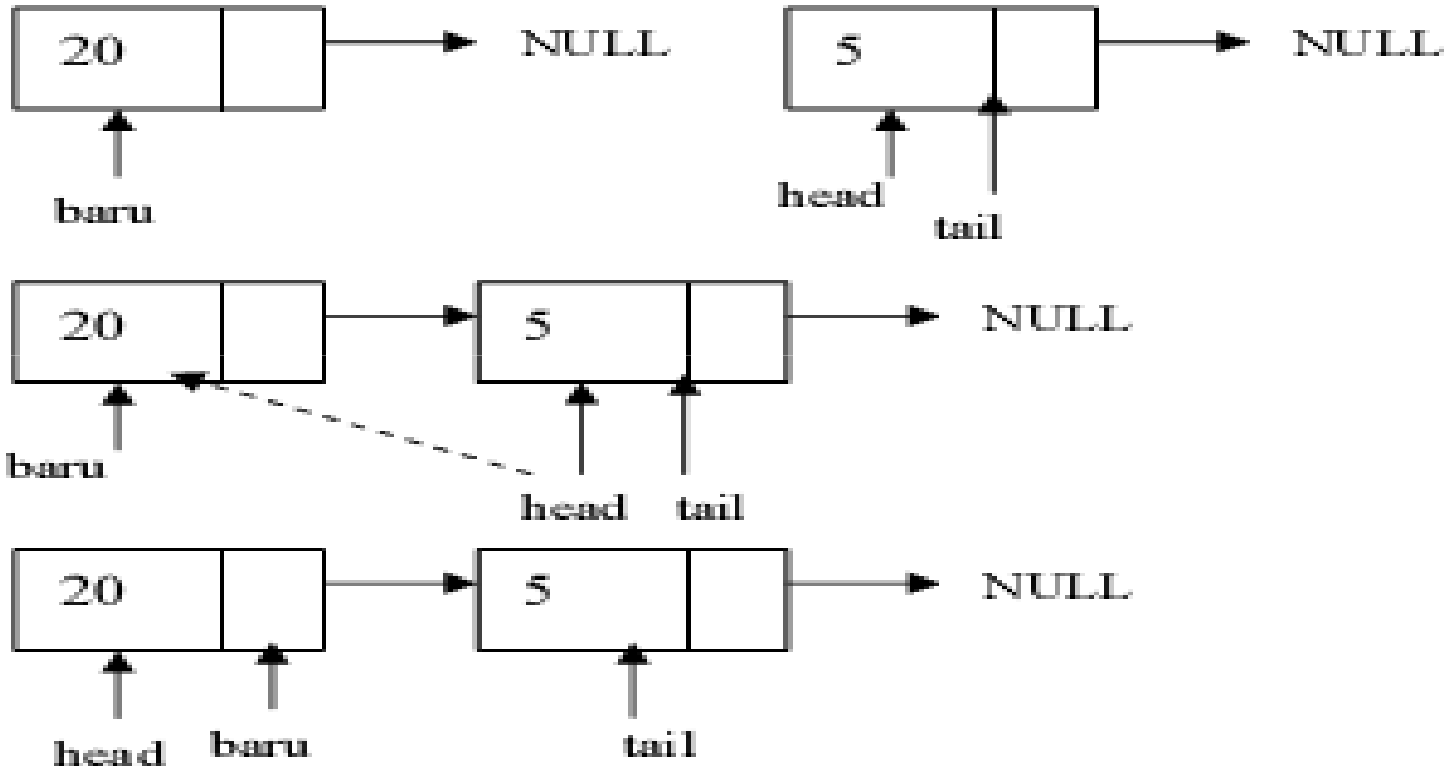
1. List masih kosong (head=tail=NULL)



2. Masuk data baru, misalnya 5



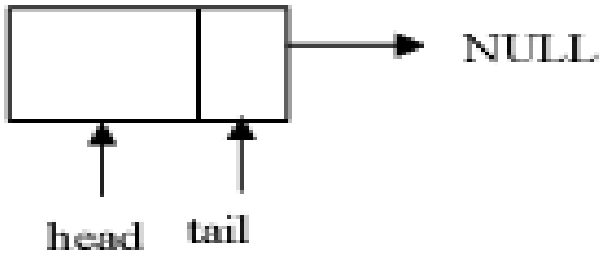
3. Datang data baru, misalnya 20



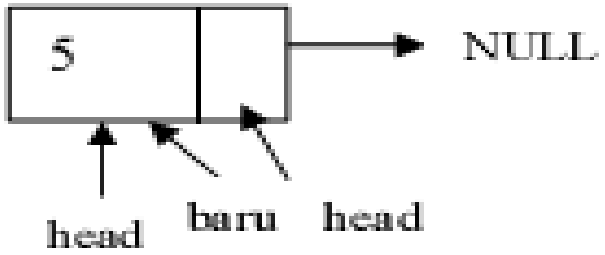
Menambah Node di Belakang Dengan Head dan Tail

```
void tambahBelakang(int databaru){
    TNode *baru,*bantu;
    baru = new TNode;
    baru->data = databaru;
    baru->next = NULL;
    if(isEmpty()==1){
     head=baru;
     tail=baru;
     tail->next = NULL;
    }
    else {
            tail->next = baru;
            tail=baru;
    }
    printf("Data masuk\n“);
}
```

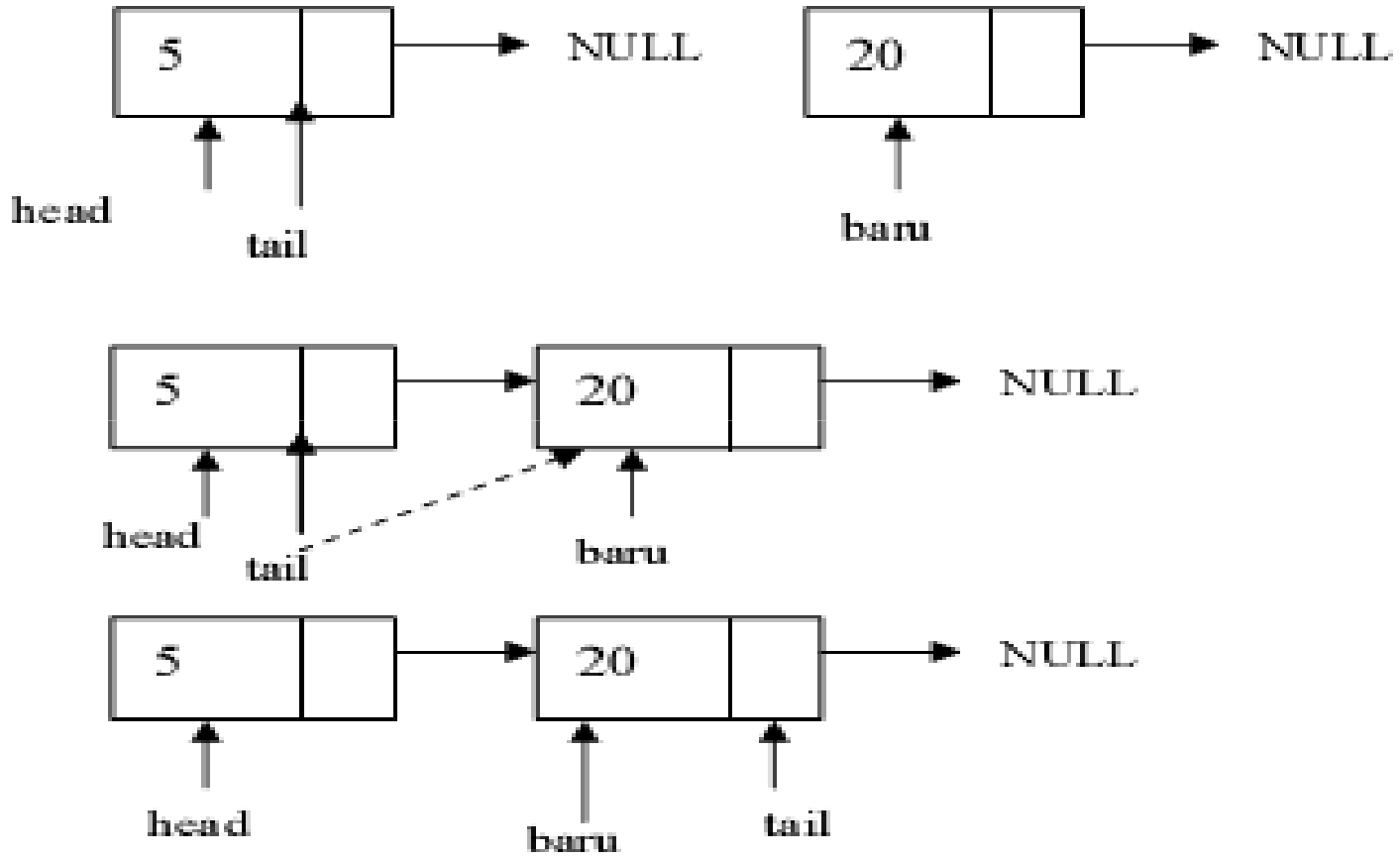
1. List masih kosong (`head=tail=NULL`)



2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20

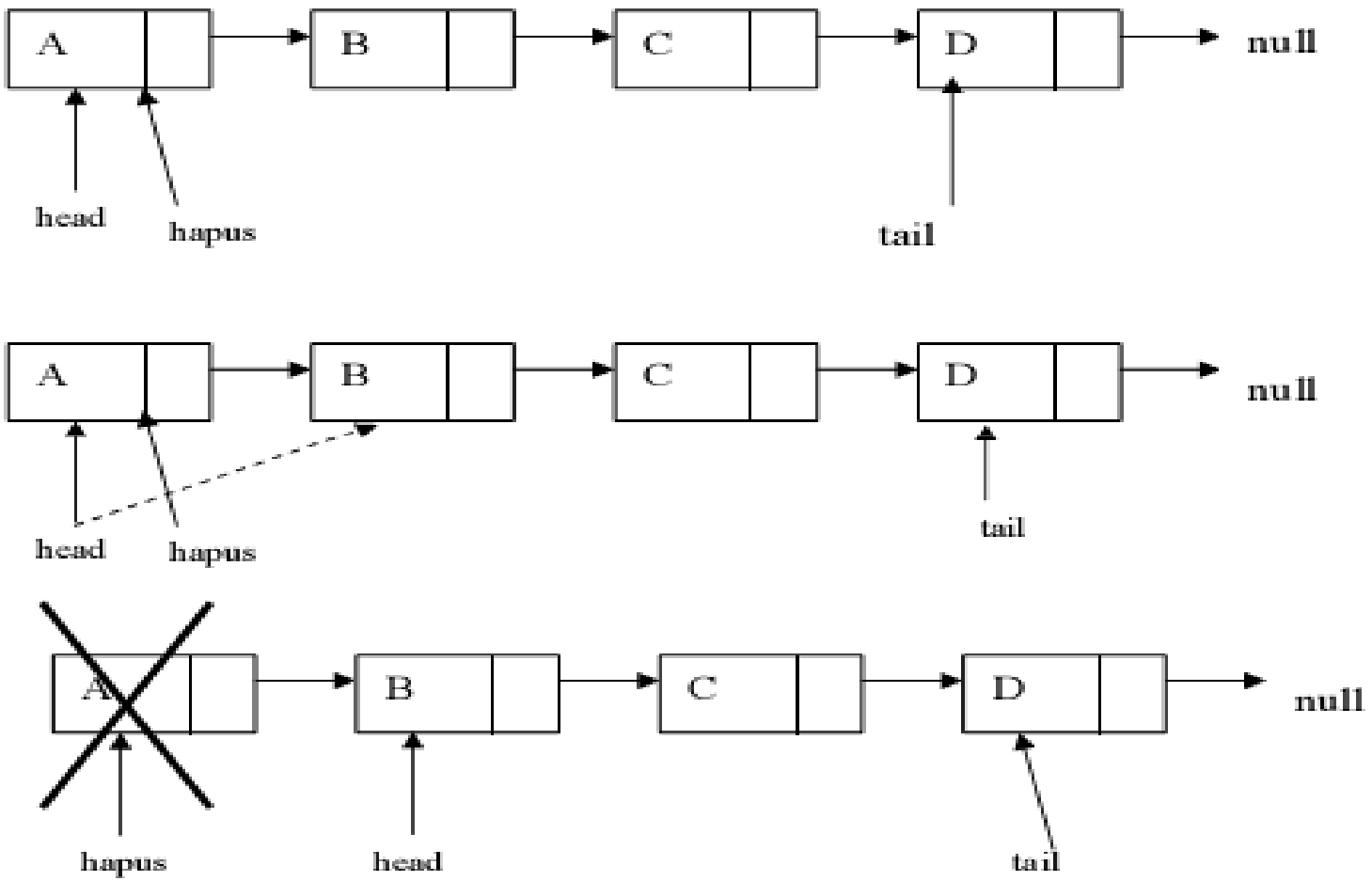


Menghapus Node di Depan (Dengan Head dan Tail)

- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penunjukkan terlebih dahulu dengan pointer hapus pada head, kemudian dilakukan pergeseran head ke node berikutnya sehingga data setelah head menjadi head baru, kemudian menghapus pointer hapus dengan menggunakan perintah delete.

- Jika tail masih NULL maka berarti list masih kosong!

```
void hapusDepan(){
    TNode *hapus;
    int d;
    if (isEmpty()==0){
          if(head!=tail){
            hapus = head;
            d = hapus->data;
            head = head->next;
            delete hapus;
          } else {
            d = tail->data;
            head=tail=NULL;
          }
          printf("%d terhapus\n",d);
    } else printf("Masih kosong\n");
}
```



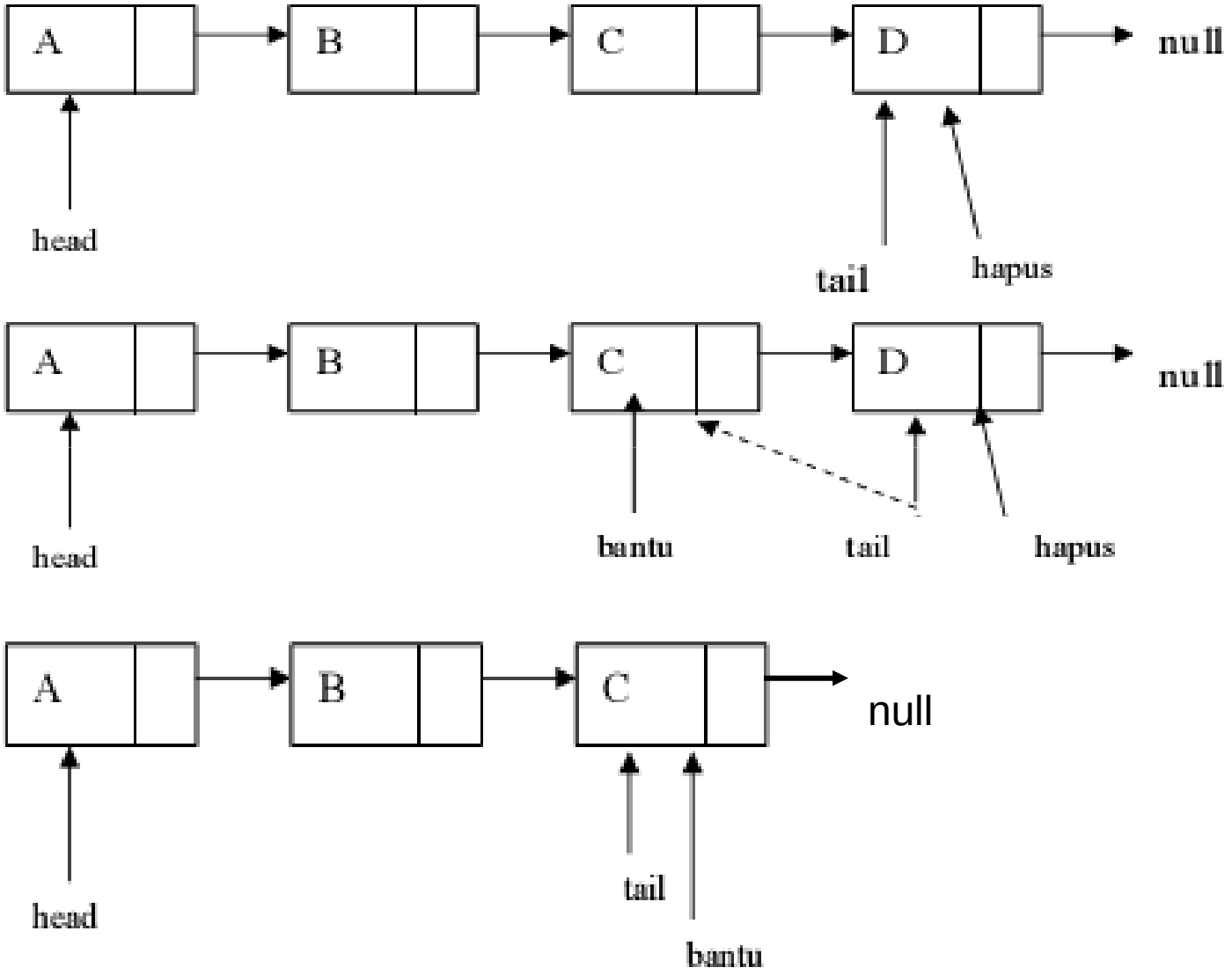
Menghapus Node di Belakang (Dengan Head dan Tail)

- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penunjukkan terlebih dahulu dengan variabel hapus pada tail. Jika tail masih NULL maka berarti list masih kosong!

- Dibutuhkan pointer bantu untuk membantu pergeseran dari head ke node berikutnya sampai sebelum tail, sehingga tail dapat ditunjukkan ke bantu, dan bantu tersebut akan menjadi tail yang baru.

- Setelah itu hapus pointer hapus dengan menggunakan perintah delete.

```
void hapusBelakang(){
    TNode *bantu,*hapus;
    int d;
    if (isEmpty()==0){
            bantu = head;
            if(head!=tail){
                    while(bantu->next!=tail){
                    bantu = bantu->next;
                    }
                    hapus = tail;
                    tail=bantu;
                    d = hapus->data;
                    delete hapus;
                    tail->next = NULL;
            }else {
                    d = tail->data;
                    head=tail=NULL;
            }
            cout<<d<<" terhapus\n";
    } else cout<<"Masih kosong\n";
}
```



Function untuk menghapus semua elemen LinkedList dengan HEAD & TAIL

```
void clear()
{
        TNode *bantu,*hapus;
        bantu = head;
        while(bantu!=NULL)
        {
           hapus = bantu;
           bantu = bantu->next;
           delete hapus;
        }
head = NULL;
tail = NULL;
}
```