DOUBLE LINKED LIST

Danang Wahyu Utomo
danang.wu@dsn.dinus.ac.id
+6285 740 955 623

RENCANA KEGIATAN PERKULIAHAN SEMESTER

W	Pokok Bahasan
1	ADT Stack
2	ADT Queue
3	List Linear
4	List Linear
5	List Linear
6	Representasi Fisik List Linear
7	Variasi List Linear
8	**Ujian Tengah Semester**

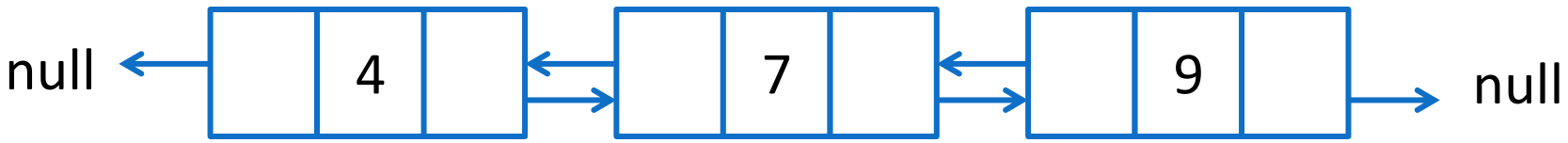
W	Pokok Bahasan
9	Variasi List Linear
10	**Double Linked List**
11	Stack dengan Representasi List
12	Queue dengan Representasi List
13	List Rekursif
14	Pohon dan Pohon Biner
15	Multi List
16	**Ujian Akhir Semester**

Double Linked List

- Memiliki dua buah pointer yaitu Pointer Prev dan Pointer Next
- Pointer Prev mengarah ke node sebelumnya
- Pointer Next mengarah ke node setelahnya

Double Linked List

- Setiap node pada linked list memiliki data dan pointer
- Inisialisasi, pointer prev dan pointer next mengarah ke NULL
- Selanjutnya, pointer prev mengarah ke node sebelumnya dan pointer next mengarah ke node setelahnya



Deklarasi dan Node Baru

- Deklarasi Node

```
typedef struct Tnode{
        int data;
        Tnode *prev;
        Tnode *next;
};
```

- Node Baru

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

Keyword new mempersiapkan
Sebuah node baru
beserta alokasi memori

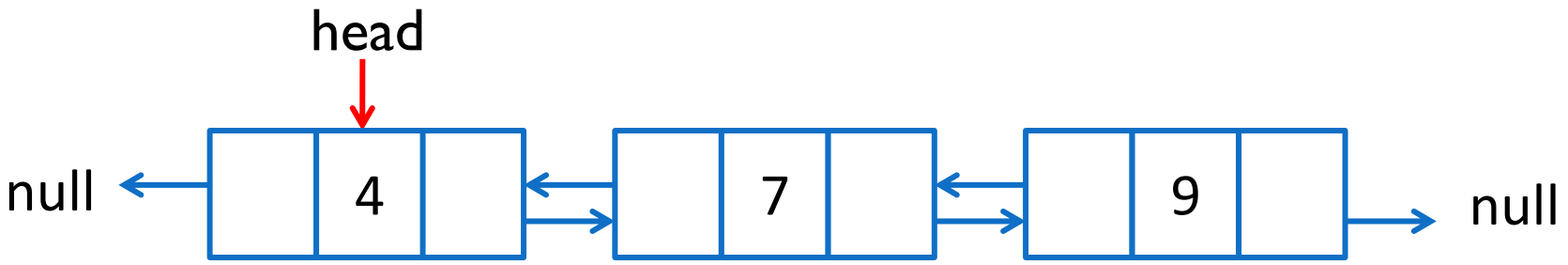
Double Linked List dengan Head

- Dibutuhkan satu buah variabel pointer : Head
- Head selalu menunjuk pada node pertama
- Manipulasi linked list harus melalui node pertama dalam linked list

```
Tnode *head;
```

- Inisialisasi :

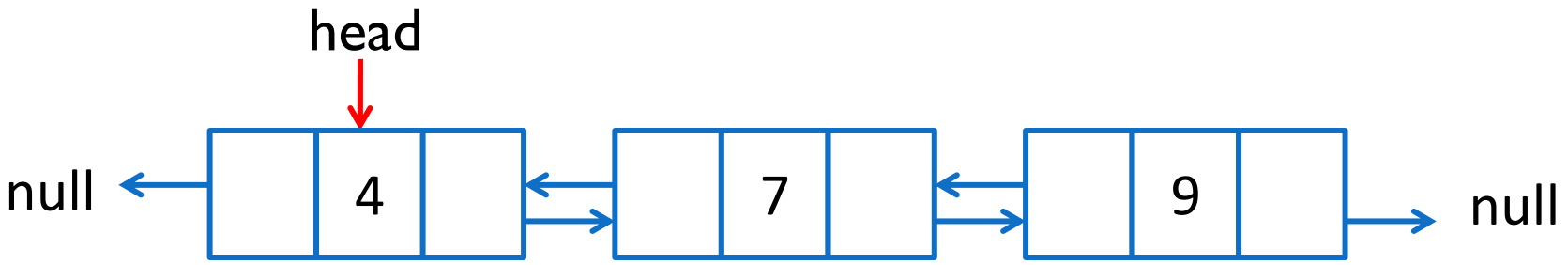
```
void init(){
     head=NULL;
}
```



Double Linked List dengan Head

- Deklarasi Pointer Penunjuk Kepala Double Linked List

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



Insert Depan

- Penambahan node baru dikaitkan pada node paling depan
- Jika masih kosong, penambahan data dilakukan pada headnya
- Prinsipnya :
  - Mengaitkan data baru dengan head
  - Head menunjuk pada data baru (head selalu menjadi data terdepan)
  - Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu

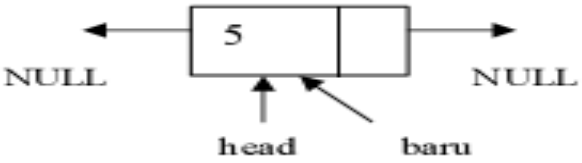
Insert Depan

```
Void insertDepan(int databaru){
        Tnode *baru;
        baru= new Tnode;
        baru->data=databaru;
        baru->prev=NULL;
        baru->next=NULL;
        if(isEmpty()==1){
                head=baru;
                head->prev=NULL;
                head->next=NULL;
        }
        else{
                baru->next = head;
                baru->prev = baru;
                head=baru;
        }
                printf("Data Masuk\n");
}
```

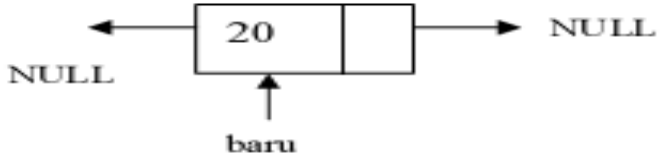
Insert Depan

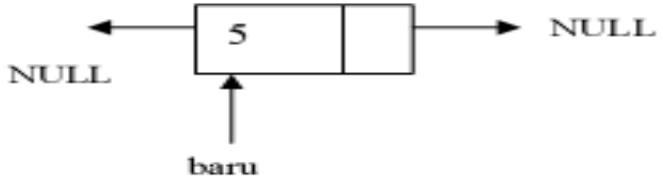
1. List masih kosong (head=NULL)

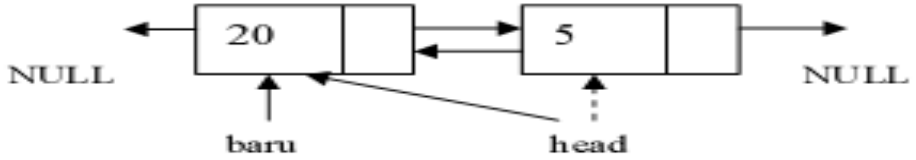
2. Masuk data baru, misalnya 5

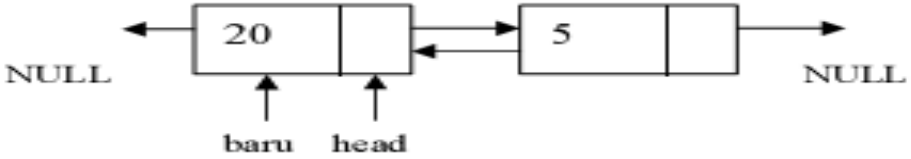


3. Datang data baru, misalnya 20









Insert Belakang

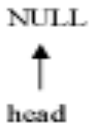
- Penambahan data dilakukan di belakang, namun pada saat pertama kali data langsung ditunjuk pada head-nya
- Penambahan di belakang lebih sulit karena membutuhkan pointer bantu untuk mengetahui data paling belakang, kemudian kaitkan dengan data baru.

Insert Belakang

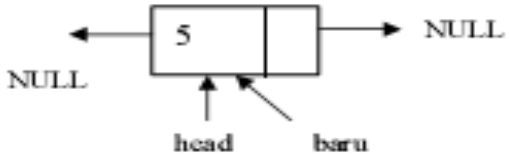
```
Void insertBelakang(int databaru){
        Tnode *baru, *bantu;
        baru = new Tnode;
        baru->data = databaru;
        baru->next = NULL;
        baru->prev = NULL;
        if(isEmpty()==1){
                head=baru;
                head->next = NULL;
                head->prev = NULL;
        }
        else{
                bantu=head;
                while(bantu->next!=NULL){
                        bantu=bantu->next;
                }
                bantu->next = baru;
                baru->prev = bantu;

        }
}
```

Insert Belakang

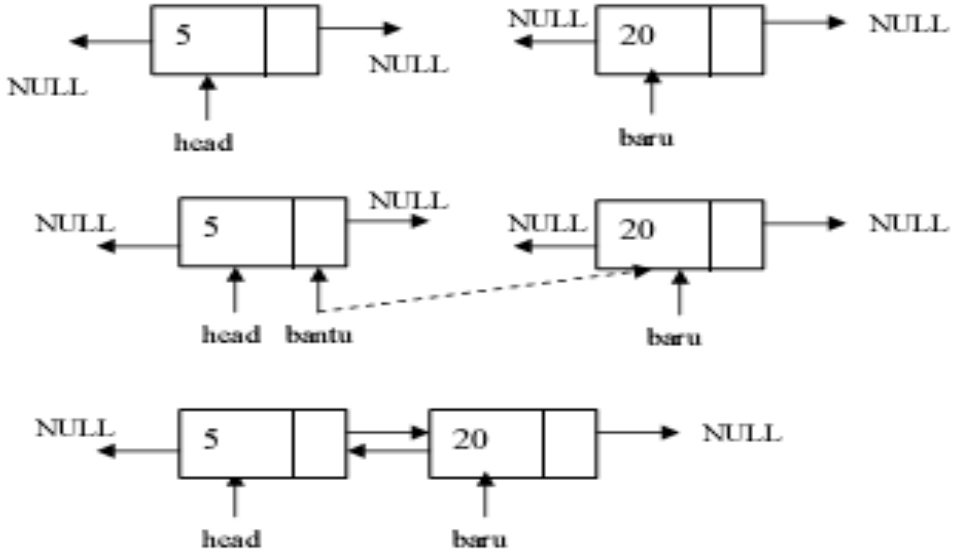
1. List masih kosong (`head=NULL`)



2. Masuk data baru, misalnya 5

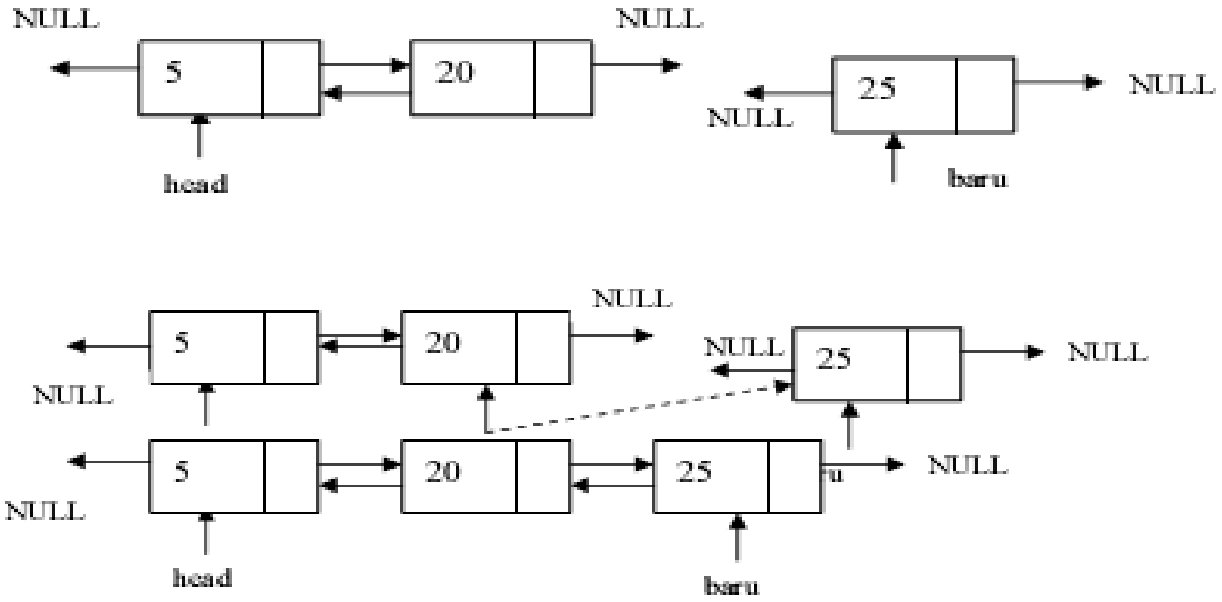


3. Datang data baru, misalnya 20 (penambahan di belakang)



Insert Belakang

4. Datang data baru, misal 25 (penambahan di belakang)



Tampil

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

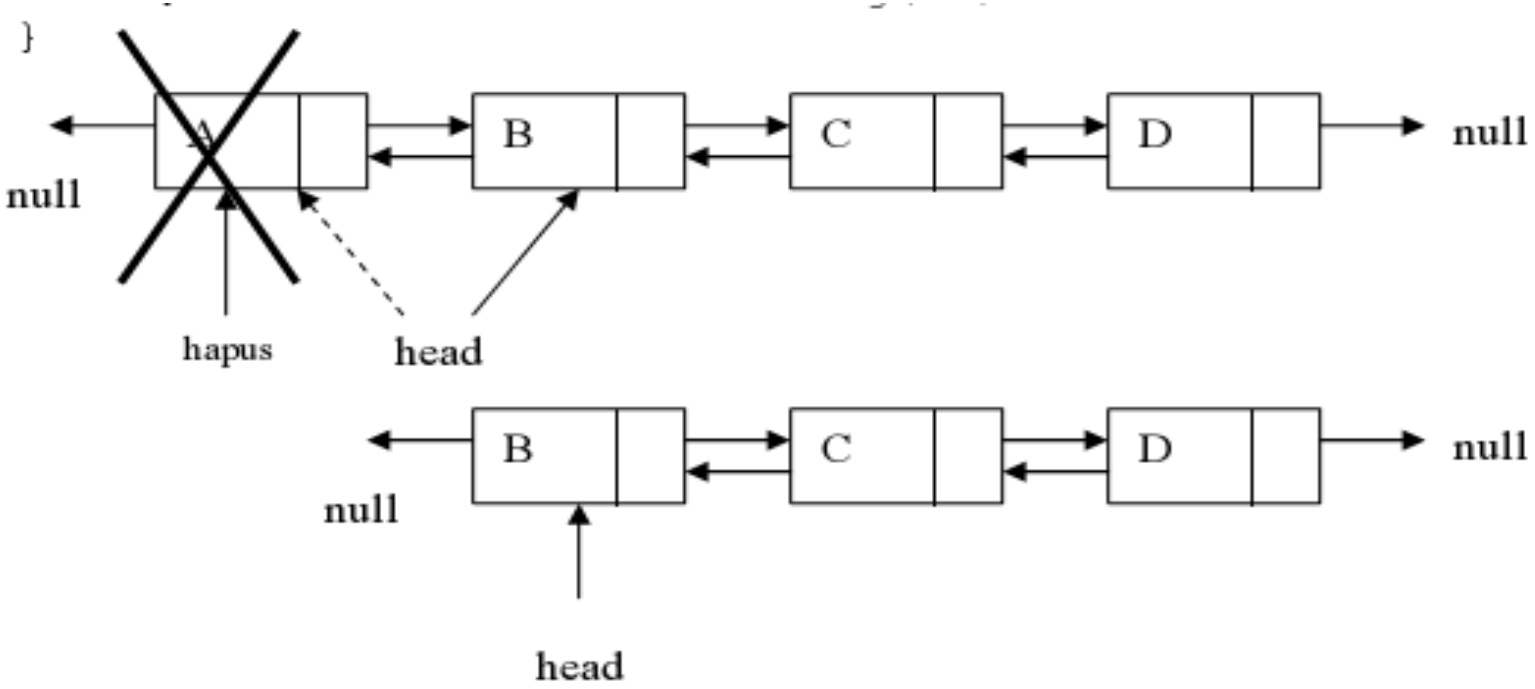
Hapus Node

- Tidak diperlukan pointer bantu yang mengikuti pointer hapus yang berguna untuk menunjuk ke NULL
- Karena pointer hapus sudah bisa menunjuk ke pointer sebelumnya dengan menggunakan elemen prev ke node sebelumnya, yang akan diset agar menunjuk ke NULL setelah penghapusan dilakukan

Hapus Node Depan

```
void hapusDepan (){
  Tnode *hapus;
  int d;
  if (isEmpty()==0){
   if(head->next != NULL){
     hapus = head;
     d = hapus->data;
     head = head->next;
           head->prev = NULL;
     delete hapus;
   } else {
     d = head->data;
     head = NULL;
   }
  }
}
```

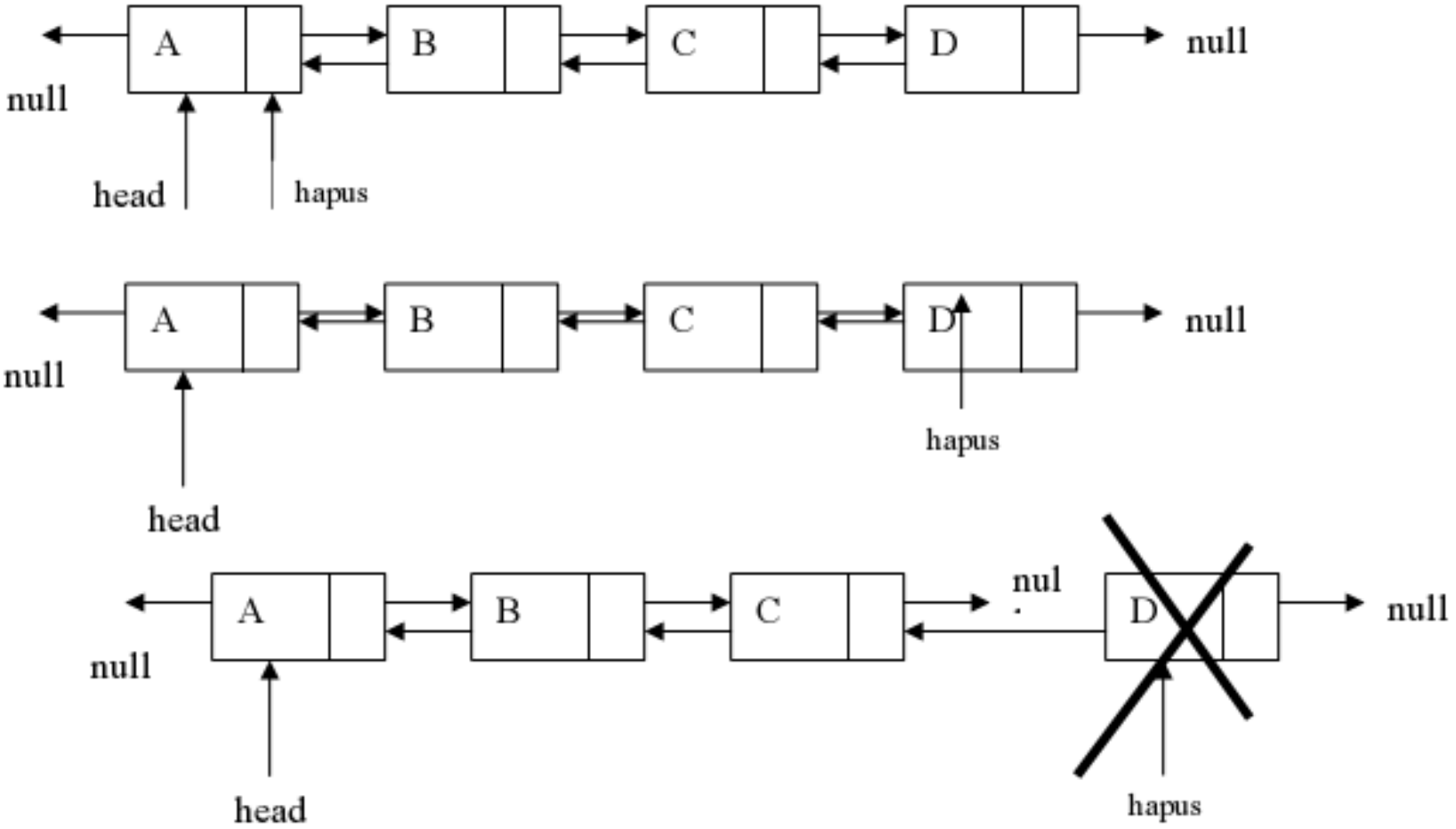
Hapus Node Depan



Hapus Node Belakang

```
void hapusBelakang(){
  Tnode *hapus;
  int d;
  if (isEmpty()==0){
   if(head->next != NULL){
     hapus = head;
     while(hapus->next!=NULL){
       hapus = hapus->next;
     }
     d = hapus->data;
     hapus->prev->next = NULL;
     delete hapus;
   } else {
     d = head->data;
     head = NULL;
   }
  }
}
```

Hapus Node Belakang

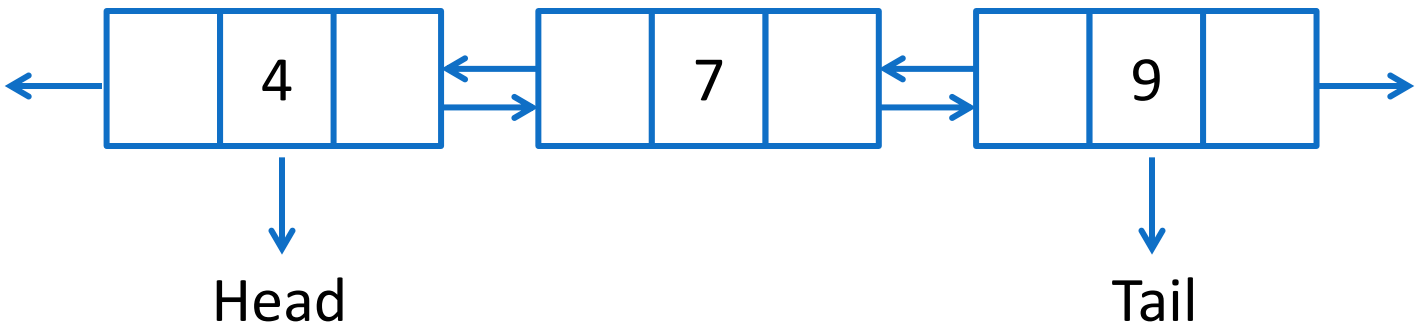


Hapus Semua Node

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

Double Linked List dengan Head & Tail

- Dibutuhkan dua buah variabel pointer : Head dan Tail
- Head akan selalu menunjuk pada node pertama
- Tail akan selalu menunjuk pada node terakhir



Double Linked List dengan Head & Tail

- **Inisialisasi**

```
Tnode *head, *tail;
```

- **Fungsi Inisialisasi Double Linked List**

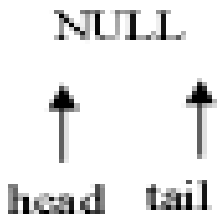
```
void init(){
      head=NULL;
      tail=NULL;
}
```

- **Fungsi untuk mengetahui list kosong**

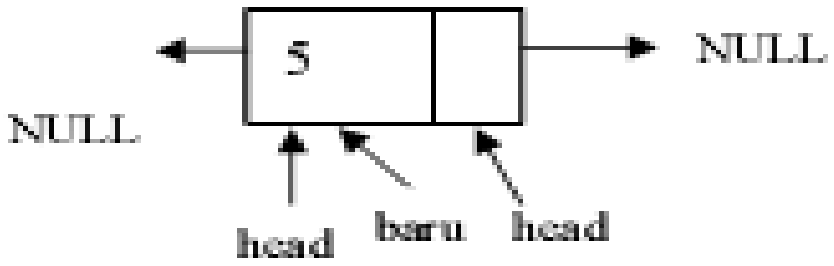
```
int isEmpty(){
      if(tail==NULL) return 1;
      else return 0;
}
```

Insert Depan dengan Head & Tail

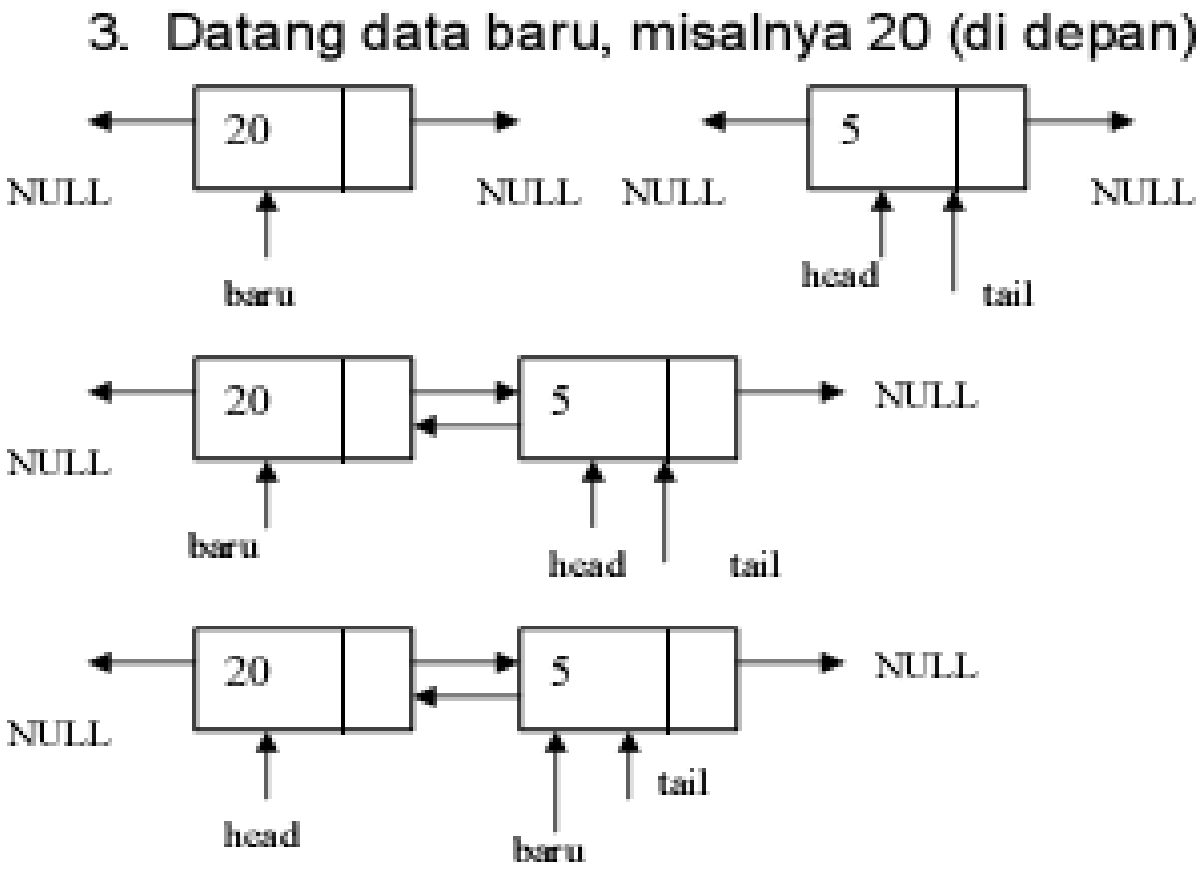
1. List masih kosong (head=tail=NULL)



2. Masuk data baru, misalnya 5



Insert Depan dengan Head & Tail



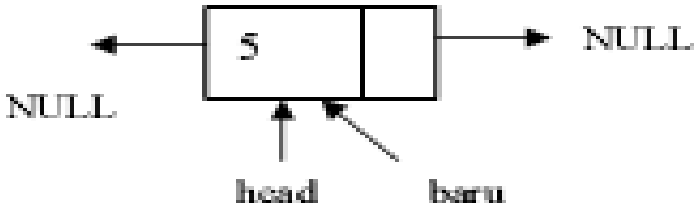
Insert Depan dengan Head & Tail

```
void insertDepan (int databaru){
  Tnode *baru;
  baru = new Tnode;
  baru->data = databaru;
  baru->next = NULL;
  baru->prev = NULL;
  if(isEmpty()==1){
       head=baru;
       tail=head;
       head->next = NULL;
       head->prev = NULL;
       tail->prev = NULL;
       tail->next = NULL;
  }
  else {
       baru->next = head;
       head->prev = baru;
       head = baru;
  }
}
```

Insert Belakang dengan Head & Tail

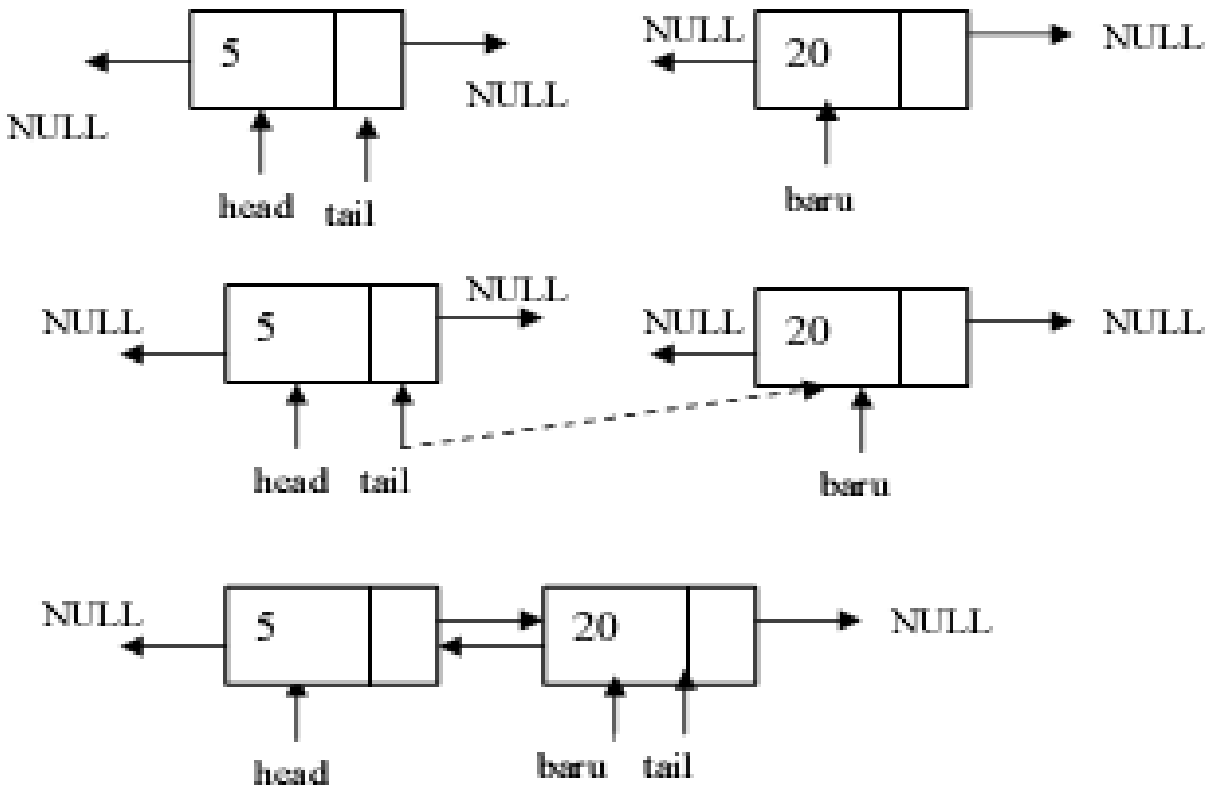
1. List masih kosong (`head=NULL`)

2. Masuk data baru, misalnya 5

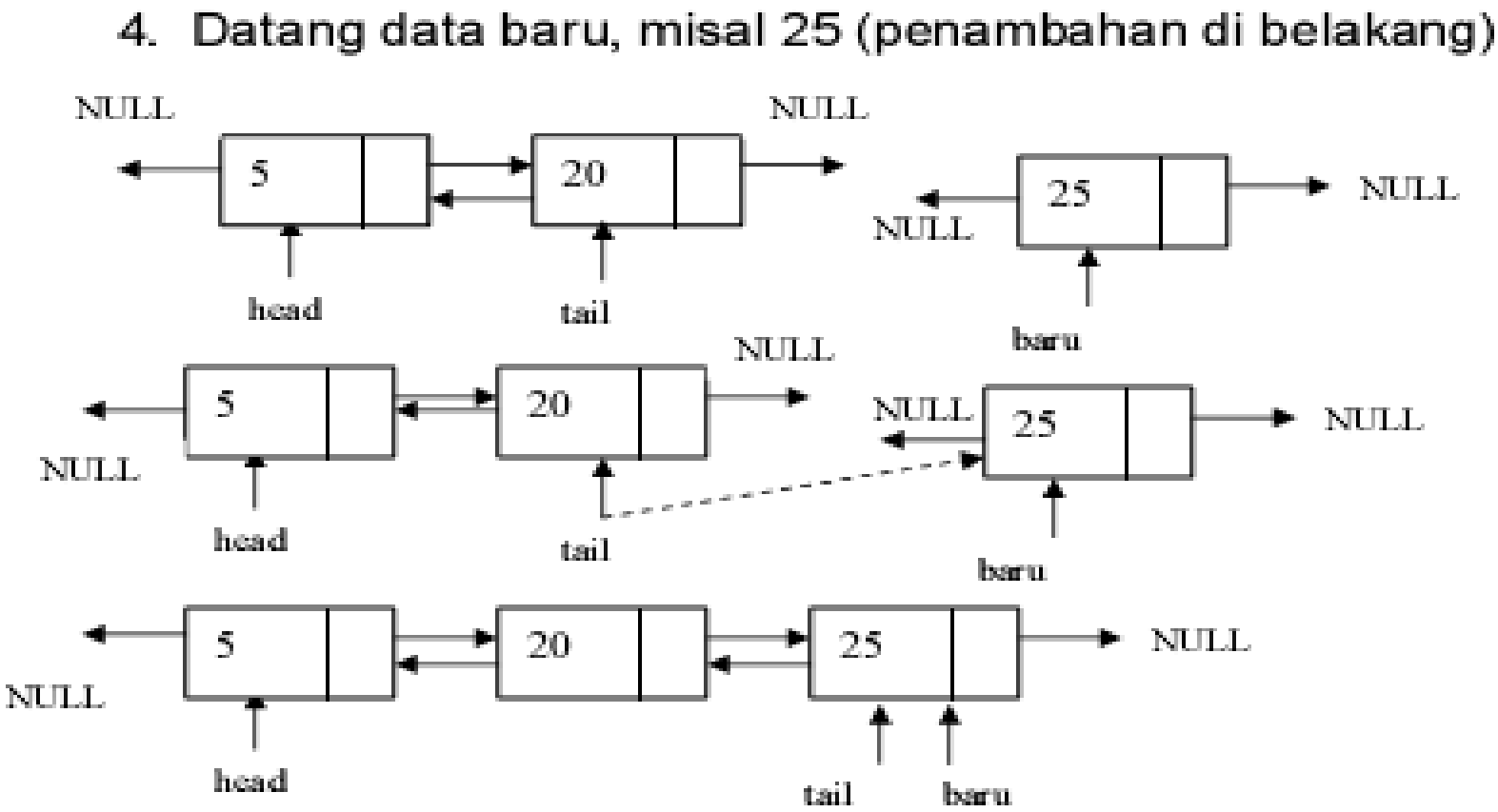


Insert Belakang dengan Head & Tail

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



Insert Belakang dengan Head & Tail



Insert Belakang dengan Head & Tail

```
void insertBelakang(int databaru){
        Tnode *baru;
        baru = new Tnode;
        baru->data = databaru;
        baru->next = NULL;
        baru->prev = NULL;
        if(isEmpty()==1){
         head=baru;
         tail=head;
         head->next = NULL;
         head->prev = NULL;
         tail->prev = NULL;
         tail->next = NULL;
        }
        else {
         tail->next = baru;
         baru->prev = tail;
         tail = baru;
         tail->next = NULL;
        }
}
```

Hapus Node Depan

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

Hapus Node Belakang

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

Hapus Semua Node

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