STIKOM BALI

LINKED LIST

Ni Kadek Sumiari, S.Kom.,M.MSI

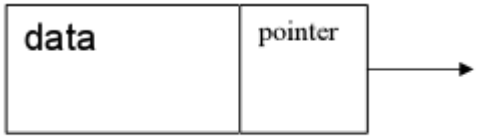
Linked List

- Linked List adalah salah satu bentuk struktur data, berisi kumpulan data (node) yang **tersusun** secara sekuensial, **saling sambung - menyambung, dinamis**.
- Linked List sering disebut juga Senarai Berantai
- Linked List saling terhubung dengan bantuan variabel pointer
- Masing-masing data dalam Linked List disebut dengan node (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa struct yang terdiri dari beberapa field.

Array VS 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

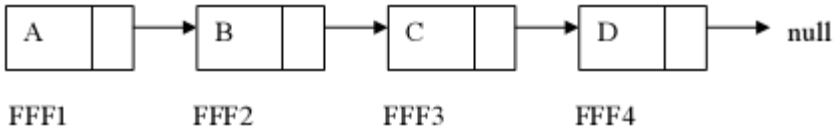
Bentuk Node Single Linked List non Circular



Menempati alamat memori tertentu

Pengertian:

- Single : artinya field pointer-nya hanya satu buah saja dan satu arah serta pada akhir node, pointernya menunjuk NULL
- Linked List : artinya node-node tersebut saling terhubung satu sama lain.



Ilustrasi Linked List

- 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 (1)

Deklarasi Node

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

Penjelasan:

- 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.

Pembuatan Single Linked List non Circular (2)

- Digunakan keyword **new** yang berarti 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;
```

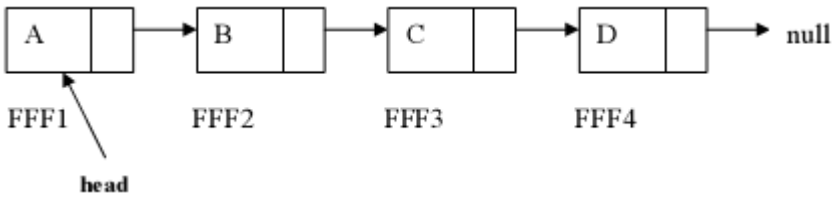
Cara lain alokasi pointer

- Menggunakan alokasi memori secara manual
- Menggunakan header **stdlib.h** atau **malloc.h**
- Menggunakan fungsi:

  <pointer type> *malloc(int size);

SLLNC MENGGUNAKAN HEAD

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



Deklarasi Pointer Penunjuk Kepala Single Linked List

- Manipulasi linked list tidak bisa dilakukan langsung ke node yang dituju, melainkan harus menggunakan suatu pointer penunjuk ke node pertama dalam linked list (dalam hal ini adalah head). Deklarasinya sebagai berikut:
- **`TNode *head;`**

SLLNC menggunakan Head

Fungsi Inisialisasi Single LinkedList


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

Function untuk mengetahui kosong tidaknya Single LinkedList

- Jika pointer head tidak menunjuk pada suatu node maka kosong

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

SLLNC dengan HEAD

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

SLLNC dengan HEAD

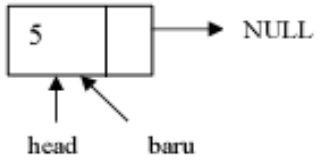
```
void insertDepan(int databaru){
  TNode *baru;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1){
    head=baru;
    head->next = NULL;
  }
  else {
    baru->next = head;
    head = baru;
  }
  printf("Data masuk\n");
}
```

SLLNC dengan HEAD

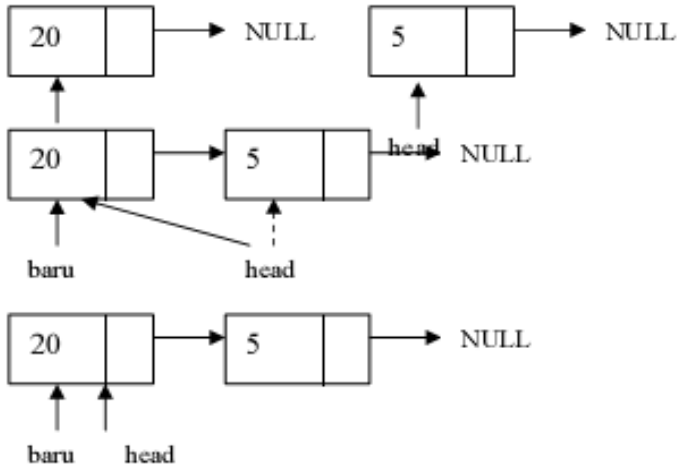
Ilustrasi:

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

2. Masuk data baru, misalnya 5



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



SLLNC dengan Head

Penambahan data di belakang

- Penambahan data dilakukan **di belakang**, namun pada saat pertama kali, node langsung ditunjuk oleh head.
- Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui node terbelakang, kemudian setelah itu, dikaitkan dengan node baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

SLLNC DENGAN HEAD

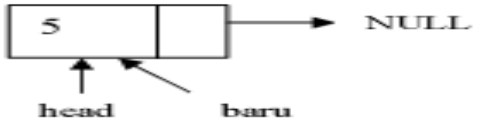
```
void insertBelakang (int databaru){
  TNode *baru,*bantu;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1){
    head=baru;
    head->next = NULL;
  }
  else {
    bantu=head;
    while(bantu->next!=NULL){
        bantu=bantu->next;
    }
    bantu->next = baru;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

SLLNC dengan HEAD

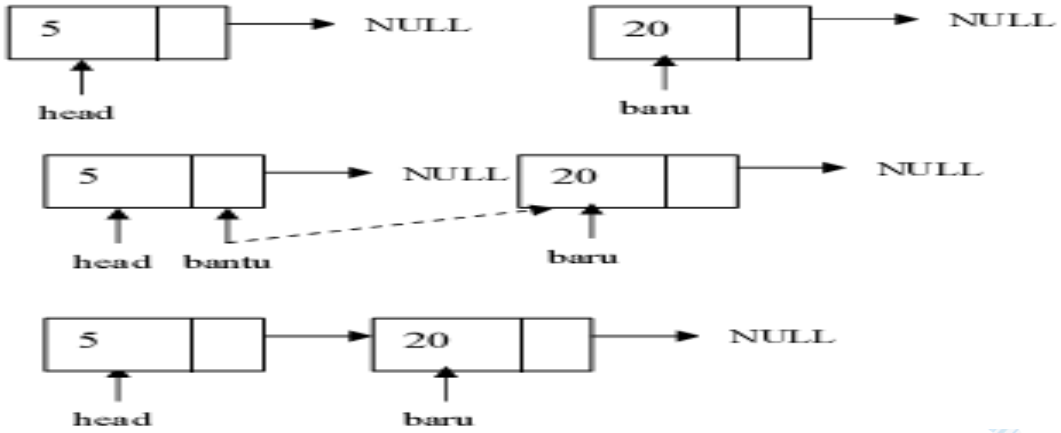
Ilustrasi:

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

2. Masuk data baru, misalnya 5

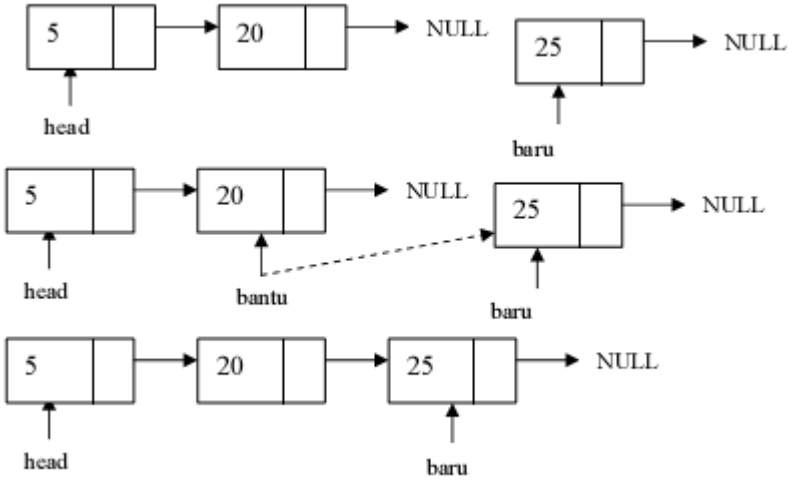


3. Datang data baru, misalnya 20 (penambahan di belakang)



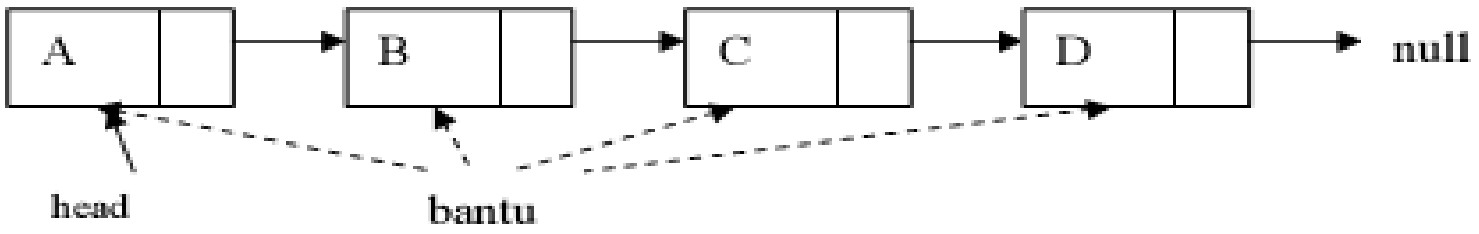
SLLNC dengan HEAD

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



SLLNC dengan HEAD

```
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");
}
```



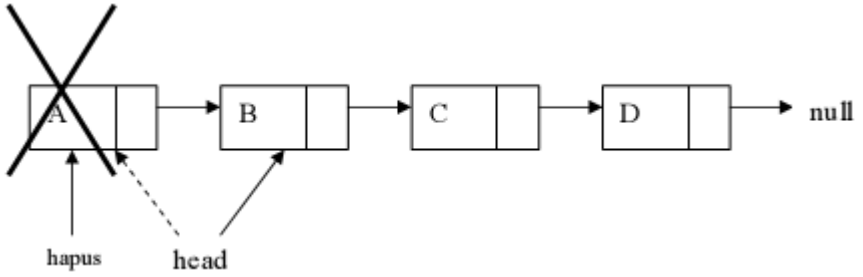
SLLNC dengan HEAD

- Function di atas digunakan untuk menampilkan semua isi list, di mana linked list ditelusuri satu-persatu dari awal node sampai akhir node. Penelusuran ini dilakukan dengan menggunakan suatu pointer bantu, karena pada prinsipnya pointer head yang menjadi tanda awal list tidak boleh berubah/berganti posisi.
- Penelusuran dilakukan terus sampai node terakhir ditemukan 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!

SLLNC dengan HEAD

Function untuk menghapus data terdepan

```
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";
}
```



SLLNC dengan HEAD

- Function di atas akan menghapus data **teratas (pertama)** yang ditunjuk oleh head pada linked list
- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penggunakan suatu pointer lain yang digunakan untuk menunjuk node yang akan dihapus, misalnya pointer hapus dan barulah kemudian menghapus pointer hapus dengan menggunakan perintah delete.
- Sebelum data terdepan dihapus, head harus ditunjukkan ke node sesudahnya terlebih dahulu agar list tidak putus, sehingga node setelah head lama akan menjadi head baru (data terdepan yang baru).
- Jika head masih NULL maka berarti data masih kosong!

SLLNC DENGAN HEAD

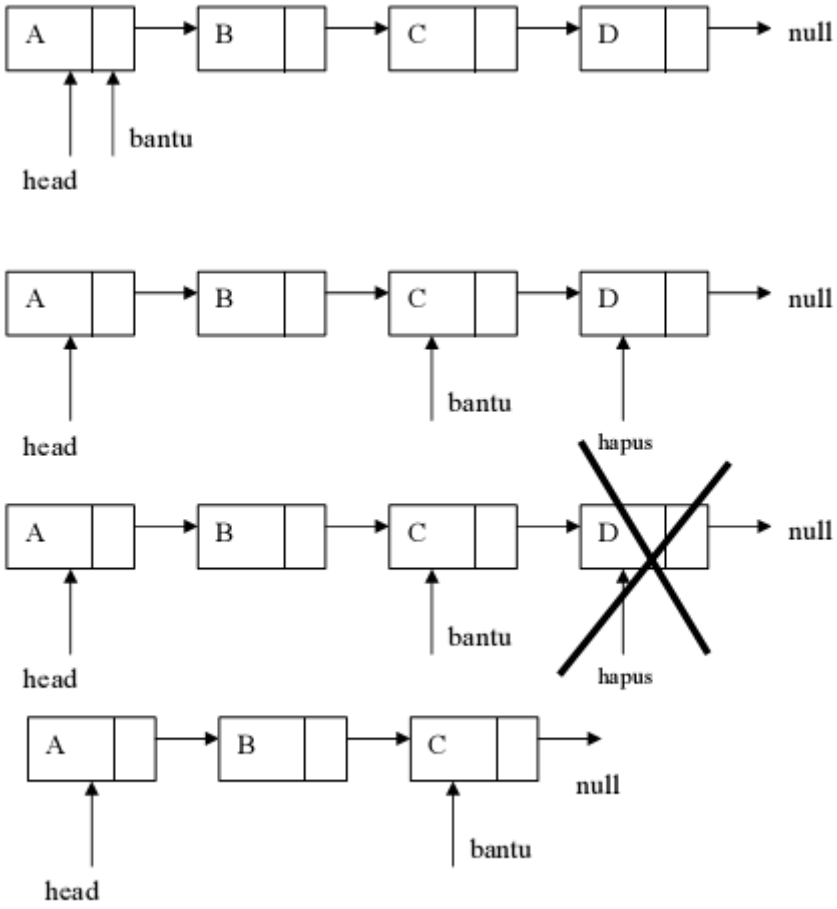
Hapus Belakang

```
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");
}
```

SLLNC dengan HEAD

- Membutuhkan pointer bantu dan hapus.
- Pointer hapus digunakan untuk menunjuk node yang akan dihapus, dan pointer bantu digunakan untuk menunjuk node sebelum node yang dihapus yang kemudian selanjutnya akan menjadi node terakhir.
- Pointer bantu akan digunakan untuk menunjuk ke nilai NULL.
- Pointer bantu akan selalu bergerak sampai sebelum node yang akan dihapus, baru kemudian pointer hapus diletakkan setelah pointer bantu. Setelah itu pointer hapus akan dihapus, pointe bantu akan menunjuk ke NULL.

SLLNC dengan HEAD



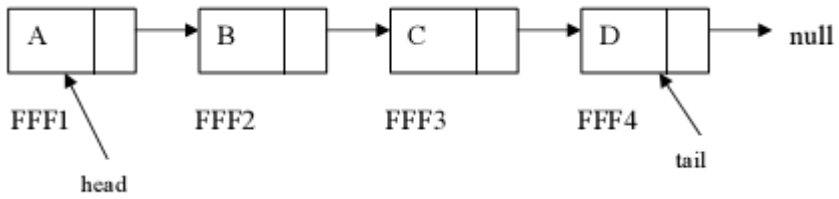
SLLNC dengan HEAD

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;
}
```

SLLNC dengan HEAD & TAIL

- Dibutuhkan dua buah variabel pointer: **head** dan **tail**
- Head akan selalu menunjuk pada node **pertama**, sedangkan tail akan selalu menunjuk pada node **terakhir**.



SLLNC dengan HEAD & TAIL

Inisialisasi LinkedList

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

Fungsi Inisialisasi LinkedList

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

Function untuk mengetahui kosong tidaknya LinkedList

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

SLLNC DENGAN HEAD & TAIL

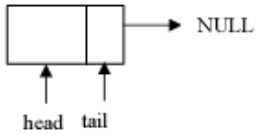
Pengkaitan node baru ke linked list di depan

Penambahan data baru di depan akan selalu menjadi head.

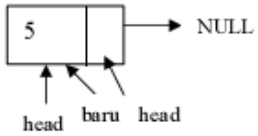
```
void insertDepan(int databaru){
  TNode *baru;
  baru = new TNode;
  baru->data = databaru;
  baru->next = NULL;
  if(isEmpty()==1){
    head=tail=baru;
    tail->next=NULL;
    }
  else {
    baru->next = head;
    head = baru;
  }
  printf("Data masuk\n");
}
```

SLLNC dengan HEAD & TAIL

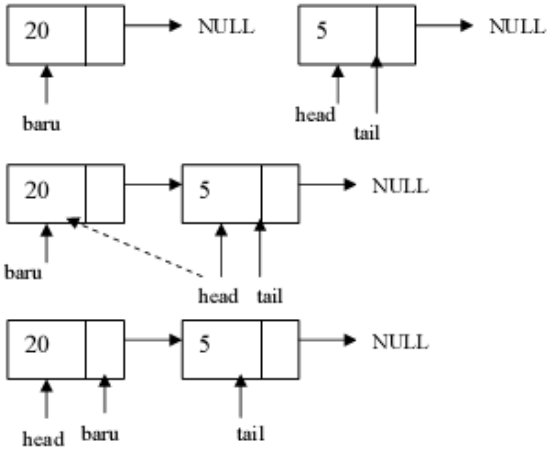
1. List masih kosong (head=tail=NULL)



2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20



SLLNC dengan HEAD & TAIL

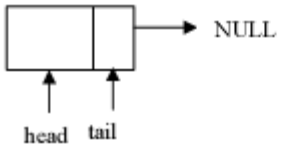
Penambahan Data di belakang

```
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");
}
```

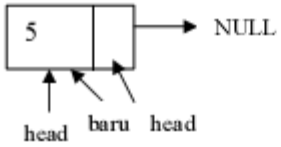
SLLNC dengan HEAD & TAIL

Ilustrasi:

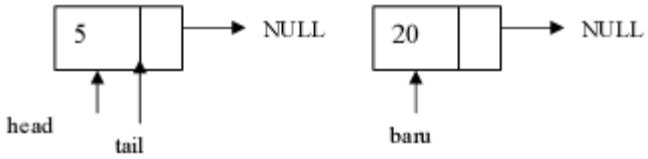
1. List masih kosong (`head=tail=NULL`)

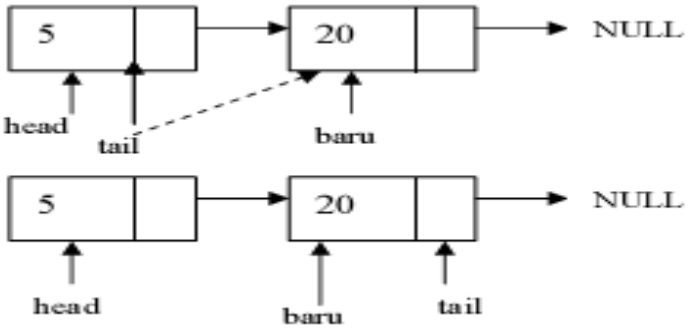


2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20



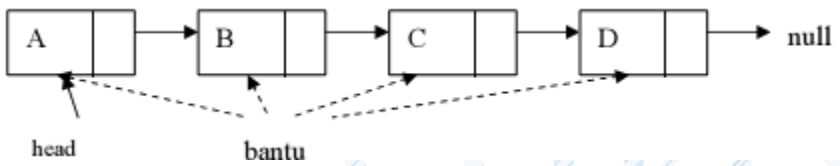


SLLNC dengan HEAD & TAIL

- 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.

Function untuk menampilkan isi linked list:

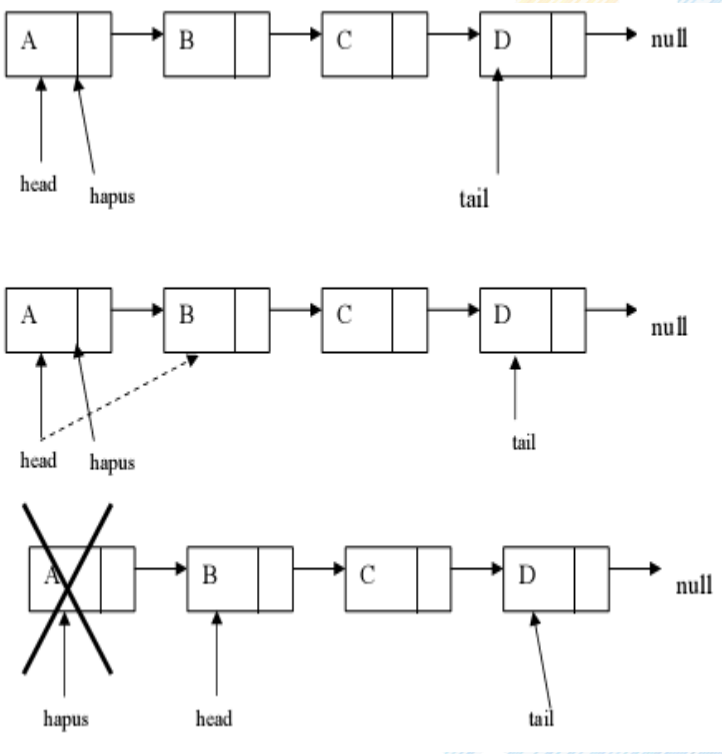
```
void tampil(){
    TNode *bantu;
    bantu = head;
    if(isEmpty()==0){
        while(bantu!=NULL){
            printf("%d\n",bantu->data);
            bantu=bantu->next;
        }
        printf("\n");
    } else printf("Masih kosong\n");
}
```



SLLNC dengan HEAD & TAIL

- Function untuk menghapus data di depan

```
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");
}
```



SLLNC DENGAN HEAD & TAIL

- Function di atas akan menghapus data **terdepan (pertama)** yang ditunjuk oleh head pada linked list
- 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!

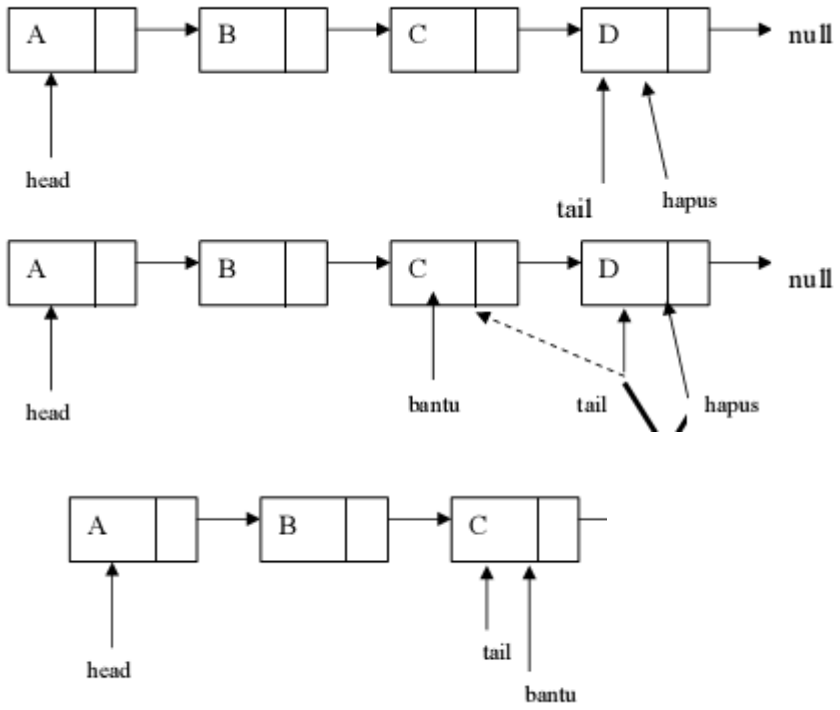
SLLNC DENGAN HEAD & TAIL

Function untuk menghapus data di belakang:

```
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";
}
```

SLLNC dengan HEAD & TAIL

Ilustrasi:



SLLNC DENGAN HEAD & TAIL

- Function di atas akan menghapus data **terbelakang (terakhir)** yang ditunjuk oleh tail pada linked list
- Penghapusan node tidak boleh dilakukan jika keadaan node sedang ditunjuk oleh pointer, maka harus dilakukan penunjukkan terlebih dahulu dengan variabel hapus pada tail, kemudian dibutuhkan pointer bantu untuk membantu pergeseran dari head ke node berikutnya sampai sebelum tail, sehingga tail dapat ditunjukkan ke bantu tersebut, dan bantu tersebut akan menjadi tail yang baru. Setelah itu hapus pointer hapus dengan menggunakan perintah delete.
- Jika tail masih NULL maka berarti list masih kosong!

SLLNC dengan HEAD & TAIL

Function untuk menghapus semua elemen LinkedList

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

Contoh Code

```
#include <iostream>
#include <memory>
#include <stdio.h>

using namespace std;

int pil;
void pilih();
void buat_baru();
void tambah_belakang();
void tambah_depan();
void tampil();
```

```
struct simpul
{
    char nim[8], nama [20];
    int umur;
    struct simpul *next;
} mhs, *baru, *awal=NULL, *akhir=NULL,*hapus,*bantu;


int main()
{
    do
    {

        cout<<"MENU SINGLE LINKEDLIST"<<endl;
        cout<<"1. Tambah Depan"<<endl;
        cout<<"2. Tambah Belakang"<<endl;
        cout<<“3. Tampilkan"<<endl;
        cout<<“4. Selesai"<<endl;
        cout<<"Pilihan Anda : ";
        cin>>pil;
        pilih();
    } while(pil!=4);
    return 0;
}
```

```
void pilih()
{
    if(pil==1)
        tambah_depan();
    else if(pil==2)
        tambah_belakang();
    else if(pil==3)
        tampil();
    else
        cout<<"selesai";
}

void buat_baru()
{
    baru=(simpul*)malloc(sizeof(struct simpul));
    cout<<"input nim   : ";cin>>baru->nim;
    cout<<"input nama  : ";cin>>baru->nama;
    cout<<"input umur  : ";cin>>baru->umur;
    baru->next=NULL;
}
```

```
void tambah_belakang()
{
    buat_baru();
    if(awal==NULL)
    {
        awal=baru;
    }
    else
    {
        akhir->next=baru;
    }
    akhir=baru;
    akhir->next=NULL;
    cout<<endl<<endl;
    tampil();
}
```

```
void tambah_depan()
{
    buat_baru();
    if(awal==NULL)
    {
        awal=baru;
        akhir=baru;
        akhir->next=NULL;
    }
    else
    {
        baru->next=awal;
        awal=baru;
    }
    cout<<endl<<endl;
    tampil();
}
```

```
void tampil()
{
    if (awal==NULL)
        cout<<"Kosong";
    else
    {
        bantu=awal;
        while(bantu!=NULL)
        {
            cout<<"nim : "<<bantu->nim;
            cout<<"  nama : "<<bantu->nama;
            cout<<"  umur : "<<bantu->umur<<endl;
            bantu=bantu->next;
        }
    }

}
```