BAB 7
DOUBLE LINKED LIST
(SENARAI BERKAIT GANDA)

1. Tujuan Instruksional Umum

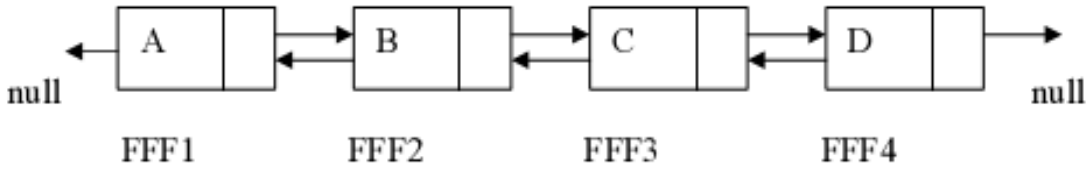
a. Mahasiswa dapat melakukan perancangan aplikasi menggunakan struktur Linked List (Senarai Berkait)
b. Mahasiswa mampu melakukan analisis pada algoritma *Linked List* yang dibuat
c. Mahasiswa mampu mengimplementasikan algoritma *Linked List* pada sebuah aplikasi secara tepat dan efisien

2. Tujuan Instruksional Khusus

a. Mahasiswa dapat menjelaskan mengenai *Linked List*
b. Mahasiswa dapat membuat dan mendeklarasikan Abstraksi Tipe Data *Linked List*
c. Mahasiswa mampu menerapkan operasi *Double Linked List Non Circular* : *Insert, update, dan delete*
d. Mahasiswa mampu menerapkan operasi *Double Linked List Circular* : *Insert, update, dan delete*

Double Linked List Non Circular

Double artinya field pointer-nya dua buah dan dua arah, ke node sebelum dan sesudahnya. Linked List artinya node-node tersebut saling terhubung satu sama lain. Non Circular artinya pointer prev dan next-nya akan menunjuk pada NULL. Jadi, Double Linked List Non Circular (DLLNC) adalah Double Linked List yang memiliki 2 buah pointer yaitu pointer next dan prev. Pointer next menunjuk pada node setelahnya dan pointer prev menunjuk pada node sebelumnya.



Gambar 1. Ilustrasi *Double Link List Non Circular*

Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke nilai NULL. Selanjutnya, pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

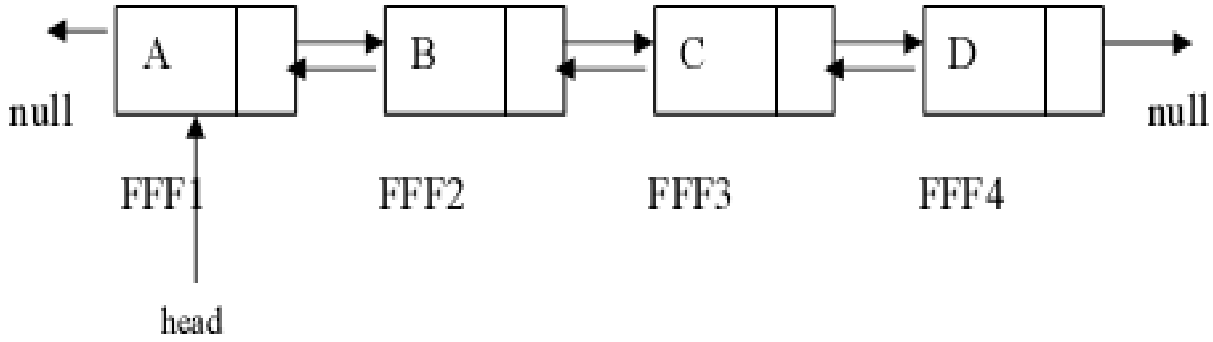
Deklarasi node dibuat dari struct berikut ini:

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

Pembentukan node baru menggunakan keyword new yang berarti mempersiapkan sebuah node baru berserta alokasi memorinya.

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

Double Linked List Non Circular dengan HEAD membutuhkan satu buah variabel pointer, yaitu: head. Head akan selalu menunjuk pada node pertama.



Gambar 2. Double Linked List Non Circular dengan HEAD

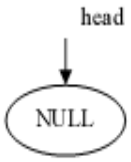
Deklarasi Pointer Penunjuk Kepala Double Linked List

Manipulasi linked list tidak bisa dilakukan langsung ke node yang dituju, melainkan harus melalui node pertama dalam linked list. Deklarasinya sebagai berikut:

```
TNode *head;
```

Fungsi Inisialisasi Single Linked List non Circular

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



Function untuk mengetahui kosong tidaknya DLLNC

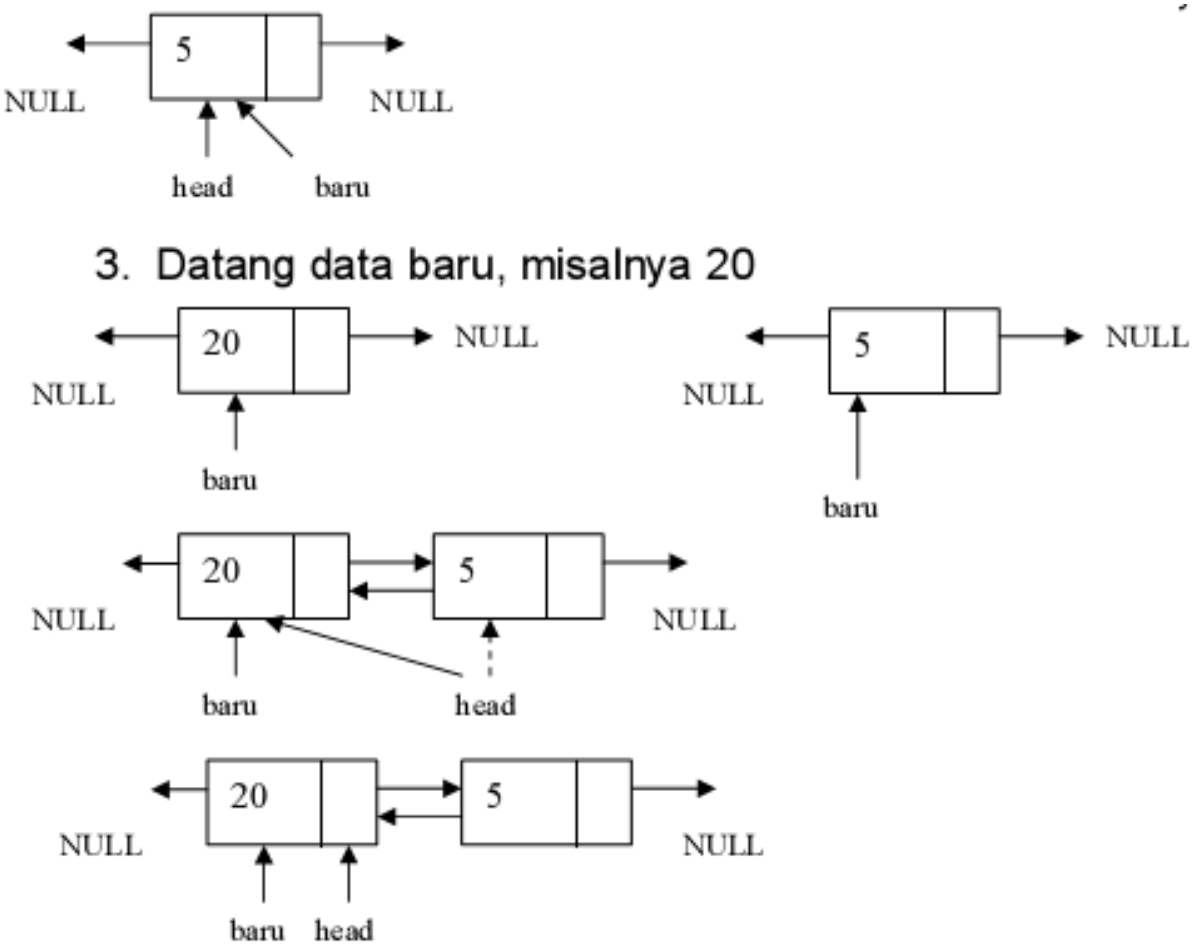
```
int isEmpty(){
     if(head == NULL) return 1;
     else return 0;
}
```

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 pada head nya. Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu.

Fungsi Menambah Di Depan

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



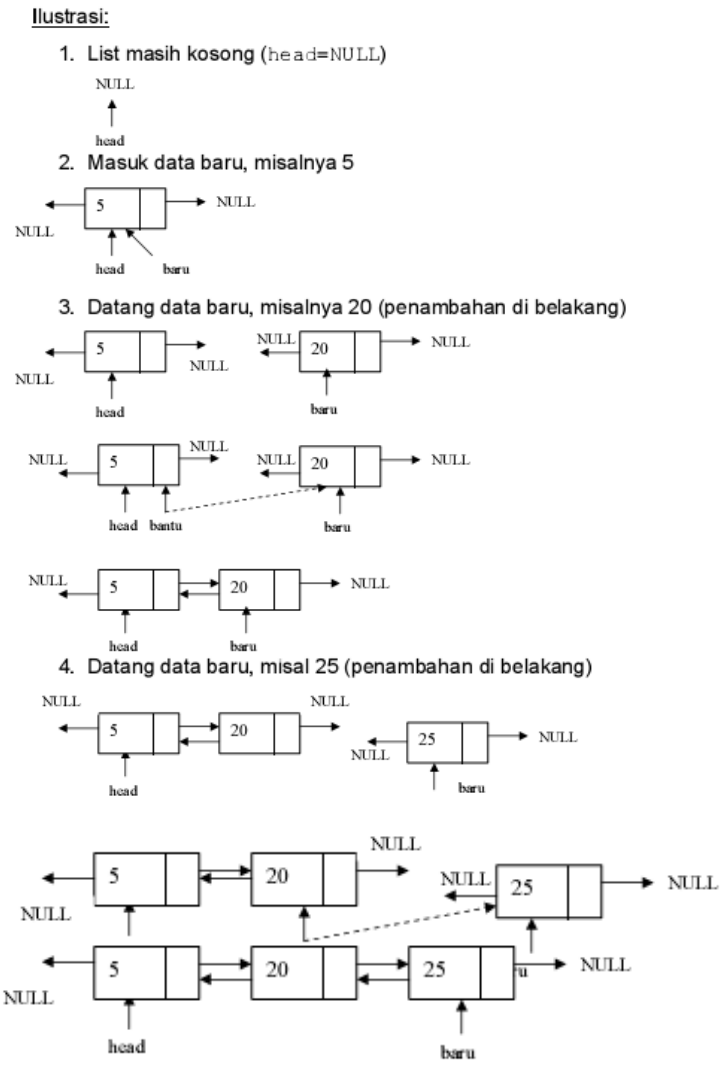
Gambar 3. Ilustrasi Penambahan Node Di Depan

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

Fungsi penambahan Node ke 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;
  }
  cout<<"Data masuk\n";
}
```

Ilustrasi Penambahan Node Di Belakang



"Bagaimana dengan penambahan di tengah?"

Function untuk menampilkan isi DLLNC

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

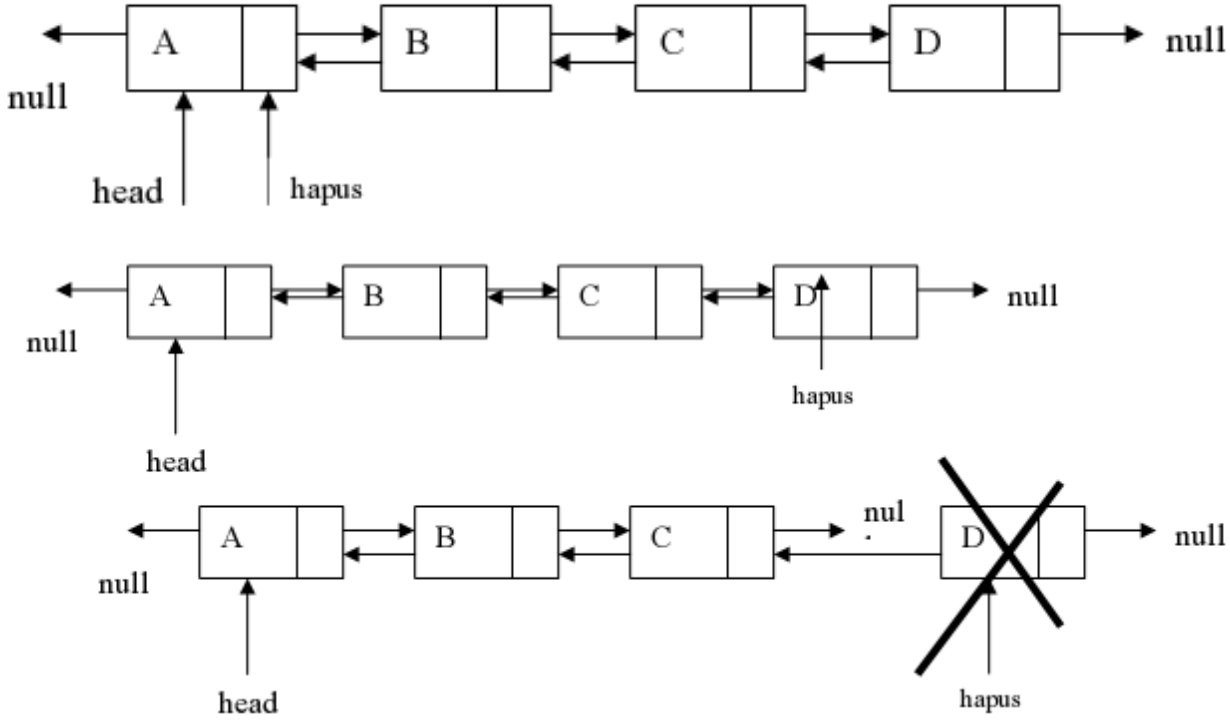
Function untuk menghapus data di depan:

```
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;
      }
      cout<<d<<" terhapus\n";
     } else cout<<"Masih kosong\n";
}
```

Fungsi untuk menghapus node terbelakang

```
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;
      }
      cout<<d<<" terhapus\n";
     } else cout<<"Masih kosong\n";
}
```

Ilustrasi Penghapusan Node



Menghapus Node Double Linked List

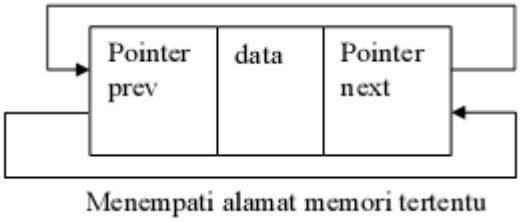
Tidak diperlukan pointer bantu yang mengikuti pointer hapus yang berguna untuk menunjuk ke NULL. Karena pointer hapus sudah bisa menunjuk ke pointer sebelumnya dengan menggunakan elemen prev ke node sebelumnya, yang akan diset agar menunjuk ke NULL setelah penghapusan dilakukan.

Fungsi menghapus semua elemen

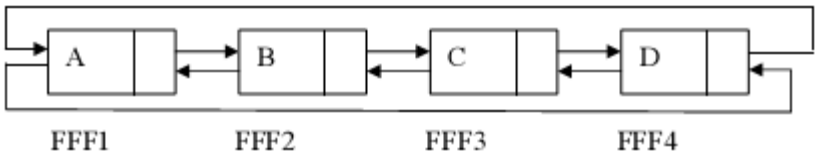
```
void clear(){
     TNode *bantu,*hapus;
     bantu = head;
     while(bantu!=NULL){
          hapus = bantu;
          bantu = bantu->next;
          delete hapus;
     }
     head = NULL;
}
```

Double Linked List Circular

Double artinya field pointer-nya terdiri dari dua buah dan dua arah, yaitu prev dan next. Linked List artinya node-node tersebut saling terhubung satu sama lain. Circular artinya pointer next dan prev-nya menunjuk ke dirinya sendiri. Jadi, Double Linked List Circular (DLLC) adalah linked list dengan menggunakan pointer, dimana setiap node memiliki 3 field, yaitu 1 field pointer yang menunjuk pointer berikutnya (next), 1 field menunjuk pointer sebelumnya (prev), serta sebuah field yang berisi data untuk node tersebut dengan pointer next dan prev nya menunjuk ke dirinya sendiri secara circular.



Ilustrasi DLLC



Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya dan ke node sebelumnya. Untuk pembentukan node baru, mulanya pointer next dan prev akan menunjuk ke dirinya sendiri. Jika sudah lebih dari satu node, maka pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node sesudahnya.

Deklarasi dan node baru DLLC

Deklarasi node

Dibuat dari struct berikut ini:

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

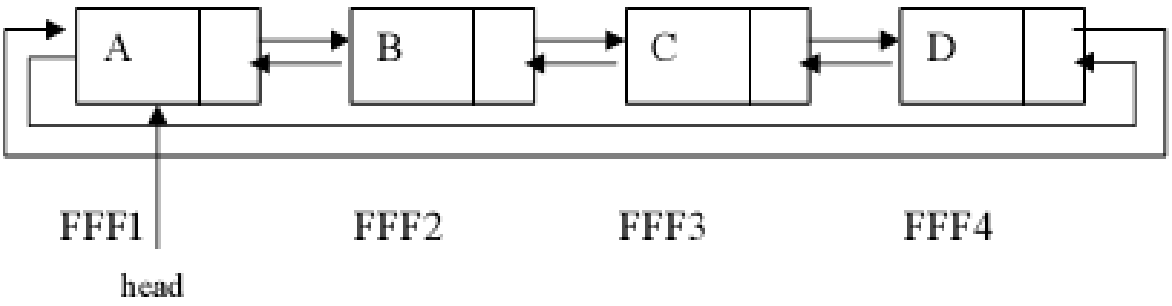
Pembentukan node baru

Digunakan keyword new yang berarti mempersiapkan sebuah node baru berserta alokasi memorinya.

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

DLLC dengan HEAD

Dibutuhkan satu buah variabel pointer, yaitu head. Head akan selalu menunjuk pada node pertama.



Deklarasi Pointer Penunjuk Kepala DLLC

Manipulasi linked list tidak bisa dilakukan langsung ke node yang dituju, melainkan harus melalui node pertama dalam linked list. Deklarasinya sebagai berikut:

```
TNode *head;
```

Fungsi Inisialisasi Single LinkedList Circular

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

Function untuk mengetahui kosong tidaknya DLLC

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

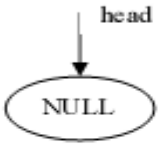
Penambahan data di depan

Penambahan node baru akan dikaitan di node **paling depan,** namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada head nya. Pada prinsipnya adalah mengkaitkan data baru dengan head, kemudian head akan menunjuk pada data baru tersebut sehingga head akan tetap selalu menjadi data terdepan. Untuk menghubungkan node terakhir dengan node terdepan dibutuhkan pointer bantu.

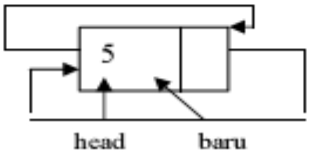
```
void insertDepan(int databaru){
  TNode *baru, *bantu;
     baru = new TNode;
     baru->data = databaru;
     baru->next = baru;
     baru->prev = baru;
  if(isEmpty()==1){
     head=baru;
     head->next = head;
     head->prev = head;
  }
  else {
     bantu = head->prev;
     baru->next = head;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

Ilustrasi:

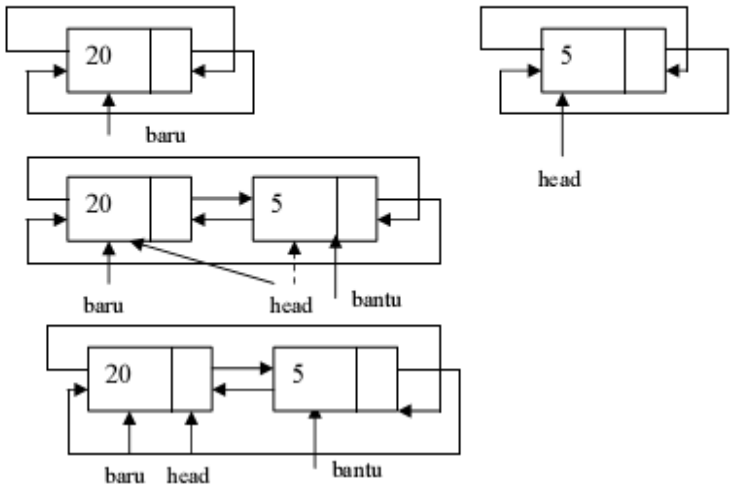
1. List masih kosong (head=NULL)



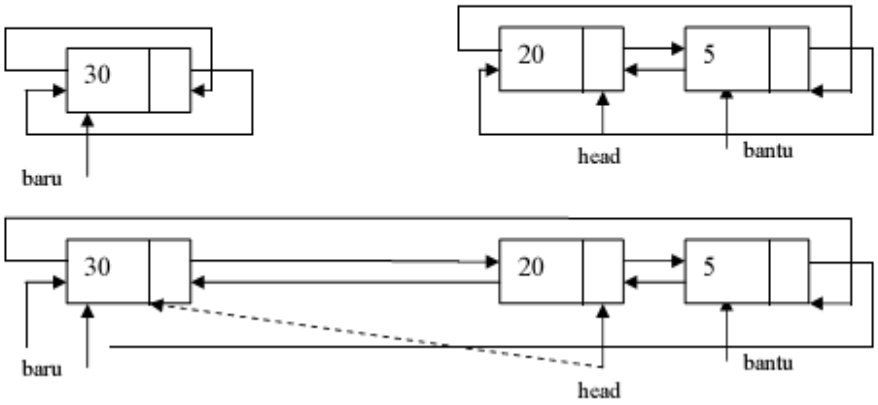
2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20



4. Datang data baru, misalnya 30



Penambahan data di belakang

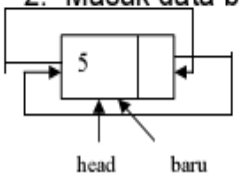
Penambahan data dilakukan **di belakang**, namun pada saat pertama kali data langsung ditunjuk pada head-nya. Penambahan di belakang lebih sulit karena kita membutuhkan pointer bantu untuk mengetahui data terbelakang, kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan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

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

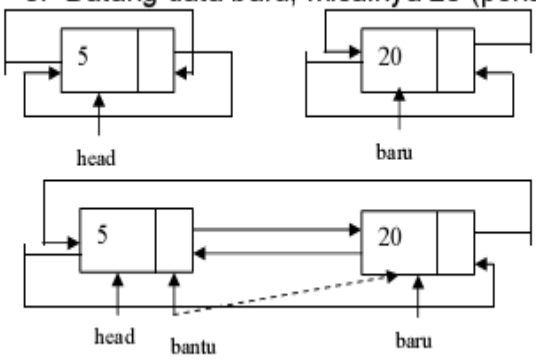
Ilustrasi:

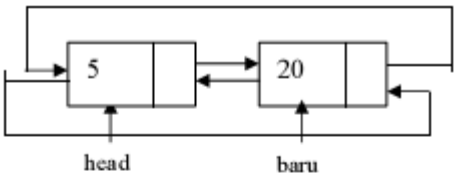
1. List masih kosong (`head=NULL`)

2. Masuk data baru, misalnya 5

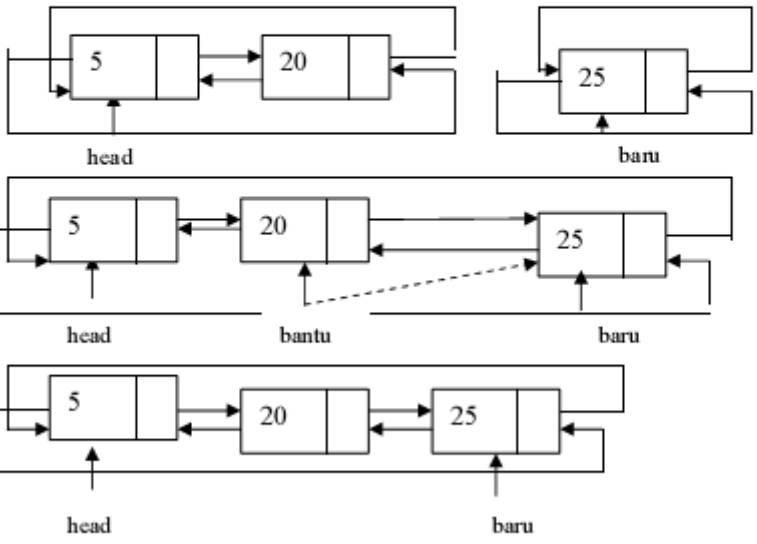


3. Datang data baru, misalnya 20 (penambahan di belakang)





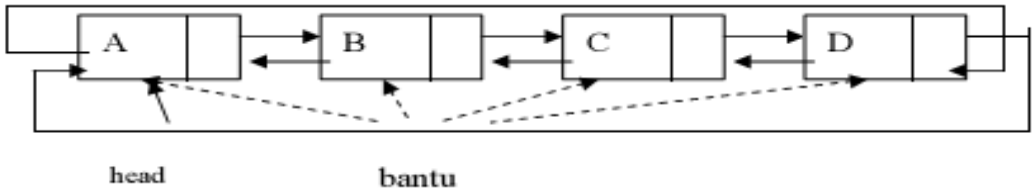
4. Datang data baru, misal 25 (penambahan di belakang)



"Bagaimana dengan penambahan di tengah?"

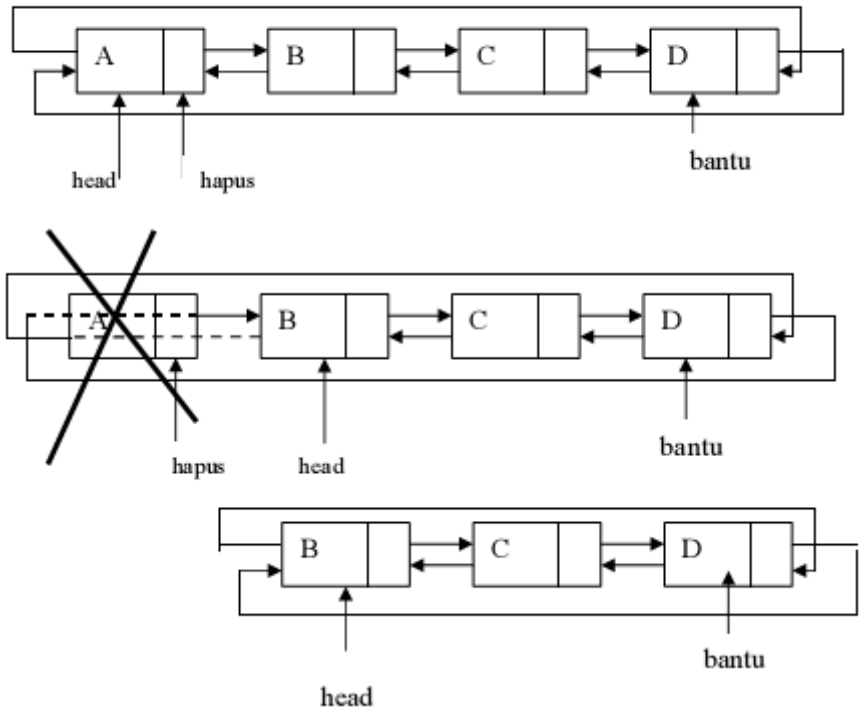
Function untuk menampilkan isi linked list

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



Function untuk menghapus node terbelakang

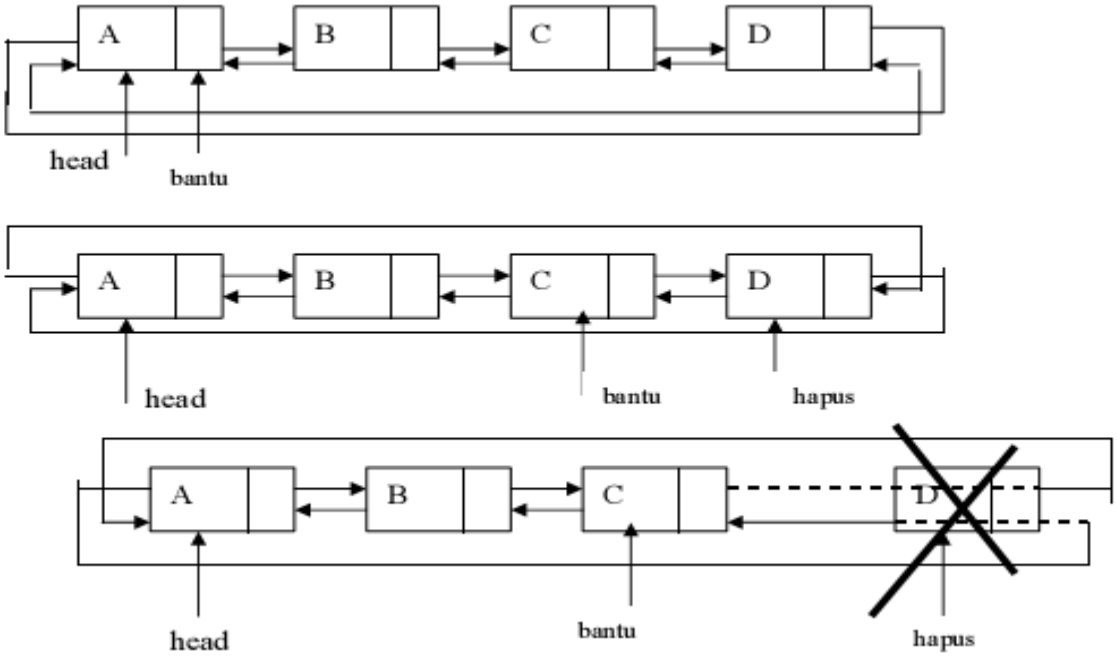
```
     void hapusDepan (){
          TNode *hapus,*bantu;
          int d;
          if (isEmpty()==0){
           if(head->next != head){
               hapus = head;
               d = hapus->data;
               bantu = head->prev;
               head = head->next;
               bantu->next = head;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



Function untuk menghapus node terbelakang

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

Diperlukan pointer bantu yang mengikuti pointer hapus yang berguna untuk menunjuk ke node sebelum terakhir. Kemudian pointer hapus ditunjukkan ke node setelah pointer bantu, kemudian hapus pointer hapus dengan perintah delete.

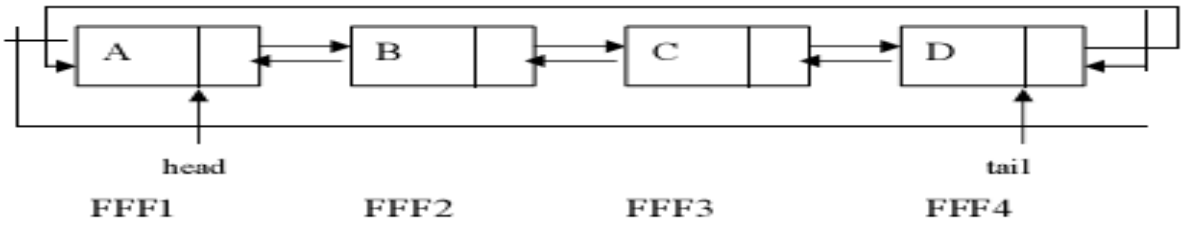


Function untuk menghapus semua elemen

```
void clear(){
     TNode *bantu,*hapus;
     if (isEmpty()==0){
          bantu = head;
          while(bantu->next!=head){
               hapus = bantu;
               bantu = bantu->next;
               delete hapus;
          }
          head = NULL;
     }
}
```

DLLC dengan HEAD dan TAIL

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



Latihan 1

```
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <malloc.h>

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

struct TNode *head;

     void insertDepan(int databaru);
     void insertBelakang(int databaru);
     void hapusDepan();
     void hapusBelakang();
     void clear();
     void tampil();

void init()
{
     head = NULL;
}

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

void insertDepan(int databaru)
{
     TNode *baru;
     baru = new TNode;
     printf("\n Data : ");
     scanf("%i",&databaru);
     printf(" Data masuk\n");

     baru->data = databaru;
     baru->next = NULL;
     baru->prev = NULL;
     if(head!=NULL)
     {
          baru->next = head;  //tambah
```

```
            head->prev = baru;
            head = baru;

      }
      else
      {
            head=baru;                //buat baru
            head->next = NULL;
            baru->prev = NULL;
      }

}


void insertBelakang (int databaru)
{
      TNode *baru,*bantu;
      baru = new TNode;
      printf("\n Data : ");
      scanf("%i",&databaru);
      printf(" Data masuk\n");
      baru->data = databaru;
      baru->next = NULL;
      baru->prev = NULL;

      if(bantu==NULL)
      {
            head=baru;                //baru
            head->next = NULL;
            head->prev = NULL;
      }
      else
      {                    //tambah
            bantu=head;
            while(bantu->next!=NULL)
            {
                  bantu=bantu->next;
            }
            bantu->next = baru;
            baru->prev = bantu;

      }

}

void tampil()
{
      TNode *bantu;
      bantu=new TNode;
```

```
     bantu = head;

     if(head!=NULL)
     {
          while(bantu!=NULL)
          {
               printf("%i",bantu->data);
               bantu=bantu->next;    printf("\n");
          }
     }
     else
          printf("\n Masih kosong\n");
}


void hapusDepan ()
{
     TNode *hapus;
     int d;

     if (head!=0)
     {
          if(head->next != NULL)
          {
               hapus = head;
               d = hapus->data;
               head = head->next;
               head->prev = NULL;
               delete hapus;
          }
          else
          {
               d = head->data;
               head = NULL;
          }
     printf("\n Terhapus\n",d);
     }
     else
          printf("\n Masih kosong\n");
}


void hapusBelakang()
{
     TNode *hapus;
     int d;

     if (head!=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;
          }
     printf("\n Terhapus\n",d);
     }
     else
          printf("\n Masih kosong\n");
}


void clear()
{
     TNode *bantu,*hapus;
     bantu = head;
     while(bantu!=NULL)
     {
          hapus = bantu;
          bantu = bantu->next;
          delete hapus;
     }
     head = NULL;
     printf("\n Semua Data Terhapus\n");
}

int main()
{
     int databaru;
     TNode *baru;
     baru = new TNode;
     baru->data = databaru;

     int pil;
     do
     {
          printf("\n\n ...Program Linked List... \n\n");
```

```
            printf(" 1. Tambah Data dari depan\n");
            printf(" 2. Tambah Data dari belakang\n");
            printf(" 3. Hapus Data dari depan\n");
            printf(" 4. Hapus Data dari belakang\n");
            printf(" 5. Clear\n");
            printf(" 6. Tampil\n");
            printf(" 7. Keluar\n");
            printf("\n Pilihan = ");
            scanf("%d",&pil);
            switch(pil)
            {
                  case 1:insertDepan(baru->data);
                        break;
                  case 2:insertBelakang(baru->data);
                        break;
                  case 3:hapusDepan();
                        break;
                  case 4:hapusBelakang();
                        break;
                  case 5:clear();
                        break;
                  case 6:tampil();
                        break;
            }
            getch();
      }
      while(pil!=8);
}
```

Latihan 2

Buatlah program Double Linked List seperti program pada Latihan 1 dengan menambahkan menu cari data.

Soal Tugas

Buatlah program Double Linked List (Non Circular atau Circular) dengan menggunakan Head untuk menyimpan seluruh elemen dalam Linked List yang mempunyai menu sebagai berikut :

1. Penambahan data di awal
2. Penambahan data di akhir
3. Penampilan data di awal
4. Penampilan data di akhir
5. Pencarian data di awal
6. Pencarian data di akhir
7. Penghapusan Data
8. Penyisipan data
9. Pengosongan data
10. Keluar dari program