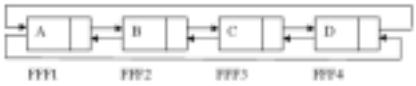
#DOUBLE LIST
Double Linked List Circular

Bahan Kuliah Struktur Data

Double Linked List Circular

- Double Linked List Circular 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.
- Double Linked List Circular pointer next dan prev nya menunjuk ke dirinya sendiri secara circular.
- **Pengertian:**
  - 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

Ilustrasi DLLC



- Setiap node pada linked list mempunyai field yang berisi data dan pointer ke node berikutnya & 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:

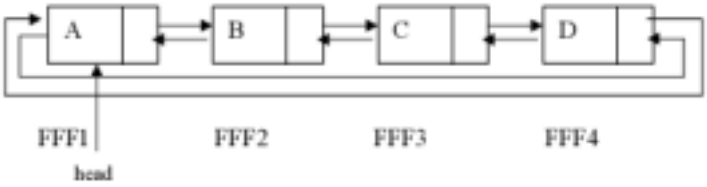
```
typedef struct TNode{
            int data;
            TNode *next;
            Tnode *prev;
};
```

- 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: head
- Head akan selalu menunjuk pada node pertama



DLLNC dengan HEAD

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;`
- `}`

DLLC dengan HEAD

Function untuk mengetahui kosong tidaknya DLLC

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

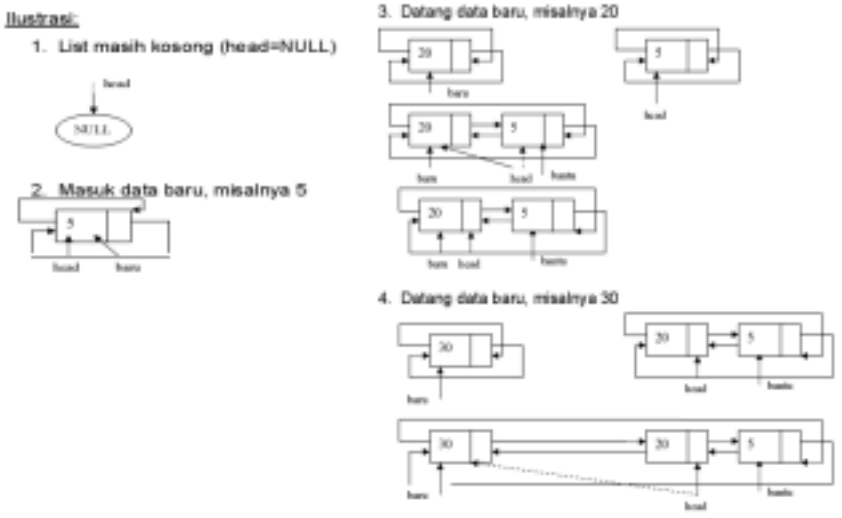
Penambahan data di depan

- 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.

DLLC menggunakan Head

```
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;
  }
  printf("Data masuk\n)";
}
```

DLLC dengan HEAD



DLLC dengan HEAD

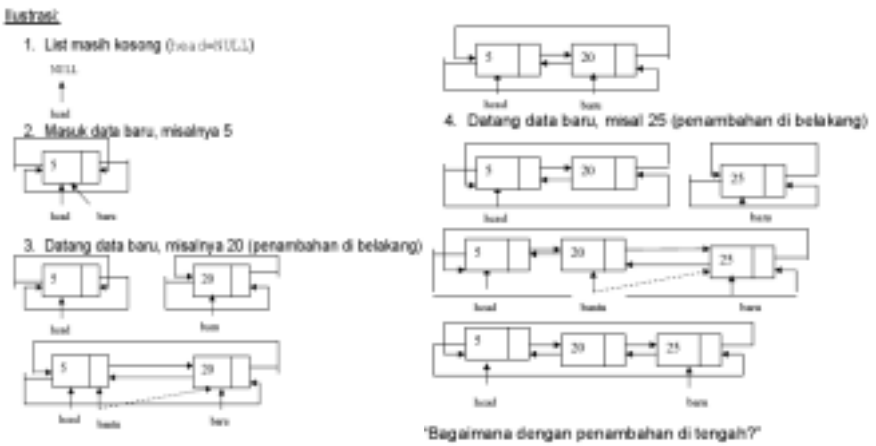
Penambahan data di belakang

- 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.

DLLC dengan Head

```
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";
}
```

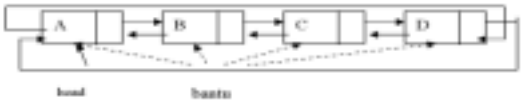
DLLC dengan HEAD



DLLC dengan HEAD

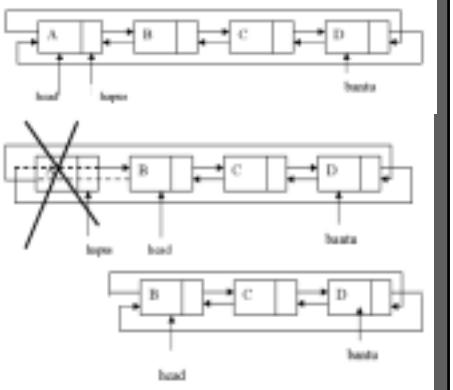
Function untuk menampilkan isi linked list

```
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";
}
```



DLLC dgn HEAD

```
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";
}
```



DLLC dengan HEAD

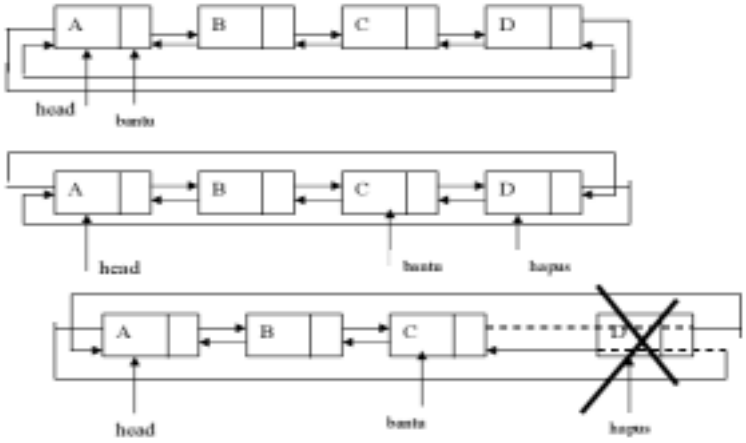
- Function untuk menghapus node terbelakang

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

DLLC dengan HEAD

- 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.

DLLC dengan HEAD



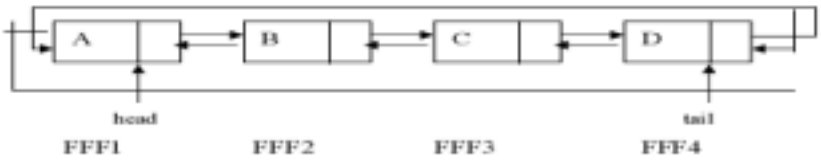
DLLC dengan HEAD

- 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: head dan tail
- Head akan selalu menunjuk pada node pertama, sedangkan tail akan selalu menunjuk pada node terakhir.



DLLC dengan HEAD dan TAIL

Inisialisasi DLLC

```
• TNode *head, *tail;
```

Fungsi Inisialisasi DLLC

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

Function untuk mengetahui kosong tidaknya DLLC

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

DLLC dengan HEAD dan TAIL

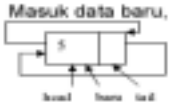
```
void insertDepan (int databaru){
  TNode *baru;
      baru = new TNode;
      baru->data = databaru;
      baru->next = baru;
      baru->prev = baru;
  if(isEmpty()==1){
      head=baru;
      tail=baru;
      head->next = head;
      head->prev = head;
      tail->next = tail;
      tail->prev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

DLLC dengan HEAD dan TAIL

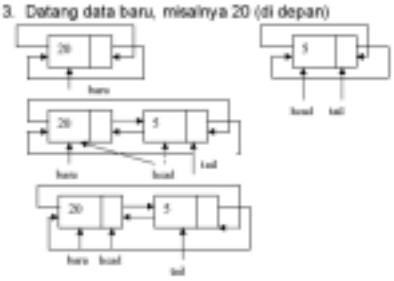
<u>Ilustrasi:</u>

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

2. Masuk data baru, misalnya 5



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



DLLC dengan HEAD & TAIL

Penambahan node di belakang

Penambahan node di belakang akan selalu dikaitkan dengan tail dan kemudian node baru tersebut akan menjadi tail

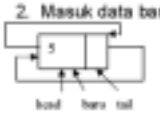
```
void insertBelakang(int databaru){
        TNode *baru;
        baru = new TNode;
        baru->data = databaru;
        baru->next = baru;
        baru->prev = baru;
        if(isEmpty()==1){
         head=baru;
         tail=baru;
         head->next = head;
         head->prev = head;
       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

DLLC dengan HEAD & TAIL

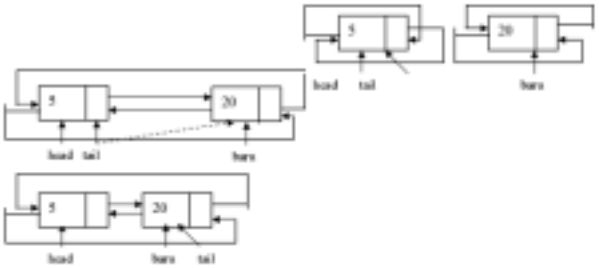
Ilustrasi:

1. List masih kosong (head=NULL)

2. Masuk data baru, misalnya 5

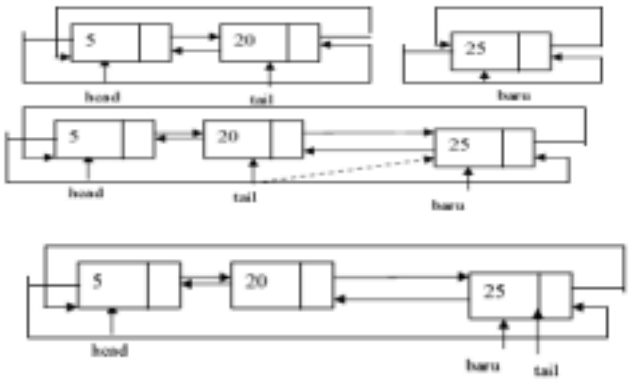


3. Datang data baru, misalnya 20 (penambahan di belakang)



DLLC dengan HEAD & TAIL

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

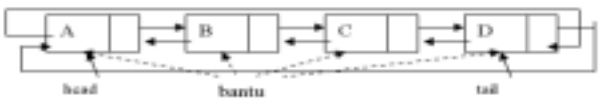


"Bagaimana dengan penambahan di tengah?"

DLLC dengan HEAD & TAIL

- Function untuk menampilkan isi linked list

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

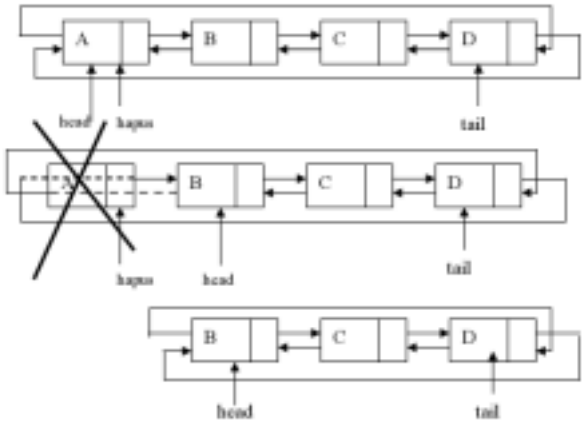


DLLC dengan HEAD & TAIL

- Function untuk menghapus data di data terdepan

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

DLLC dengan HEAD & TAIL



DLLC dengan HEAD & TAIL

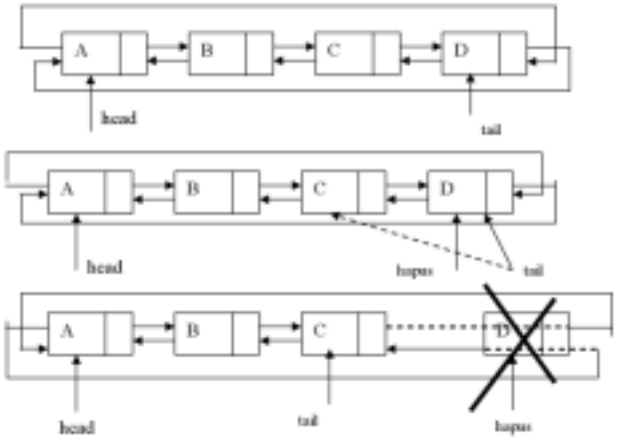
- Function untuk menghapus node terbelakang

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

DLLC dengan HEAD & TAIL

- Pointer hapus tidak perlu di loop untuk mencari node terakhir. Pointer hapus hanya perlu menunjuk pada pointer tail saja.
- Karena pointer hapus sudah bisa menunjuk ke pointer sebelumnya dengan menggunakan elemen prev ke node sebelumnya. Kemudian pointer tail akan berpindah ke node sebelumnya.

DLLC dengan HEAD & TAIL



DLLC dengan HEAD & TAIL

- **Function untuk menghapus semua elemen LinkedList**

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

DLLNC dengan HEAD & TAIL

- Menggunakan pointer bantu yang digunakan untuk bergerak sepanjang list, dan menggunakan pointer hapus yang digunakan untuk menunjuk node-node yang akan dihapus.
- Pada saat pointer hapus menunjuk pada node yang akan dihapus, pointer bantu akan bergerak ke node selanjutnya, dan kemudian pointer hapus akan didelete.

SOAL-SOAL

- Buatlah program lengkap dari semua algoritma dan function di atas dalam bentuk menu untuk menambah data, melihat data, dan menghapus data!
- Buatlah function tambahan yang berguna untuk mencari data yang ada dalam linked list baik dengan head maupun head & tail!
- Buatlah function untuk menghapus data tertentu dalam linked list!
- Buatlah function untuk menampilkan data secara terbalik!