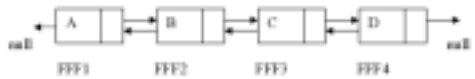
#DOUBLE LIST
Double Linked List Non Circular

Bahan Kuliah Struktur Data

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.
- **Pengertian:**
  - 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.

Ilustrasi DLLNC



- 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 nilai NULL.
- Selanjutnya pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node selanjutnya pada list.

Deklarasi dan node baru DLLNC

- Deklarasi node
- Dibuat dari struct berikut ini:

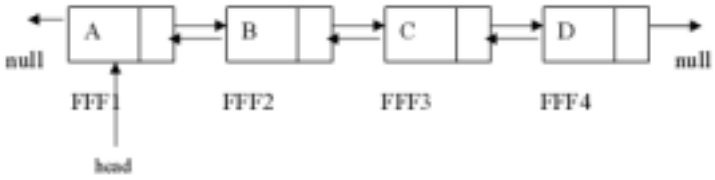
```
typedef struct TNode{
              int data;
              TNode *next;
              Tnode *prev;
};
```

- 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 = NULL;
baru->prev = NULL;
```

DLLNC dengan HEAD

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



DLLNC 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 LinkedList Circular

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

DLLNC dengan HEAD

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.

DLLNC menggunakan Head

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

DLLNC dengan HEAD

DLLNC 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.

DLLNC dengan Head

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

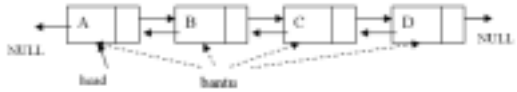
DLLNC dengan HEAD

DLLNC dengan HEAD

Function untuk menampilkan isi DLLNC

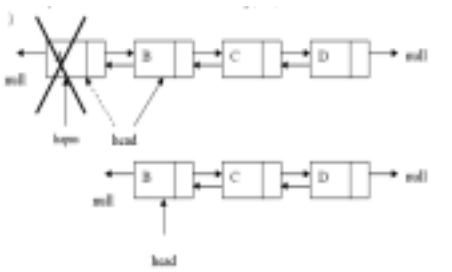
```
void tampil(){
  TNode *bantu;
  bantu = head;
  if(isEmpty()==0){
       while(bantu!=NULL){
              printf("%d ",bantu->data);
              bantu=bantu->next;
       }
       printf("\n");
  } else printf("Masih kosong\n");
}
```

Bagaimana cara membaca data list secara terbalik?



DLLNC dgn HEAD

```
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;
   }
   printf("%d terhapus\n",d);
  } else printf("Masih kosong\n");
}
```



DLLNC dengan HEAD

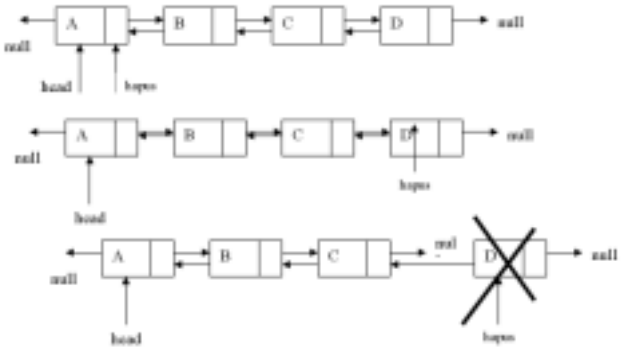
Function untuk menghapus node terbelakang

```
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";
}
```

DLLNC dengan HEAD

- 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.

DLLNC dengan HEAD



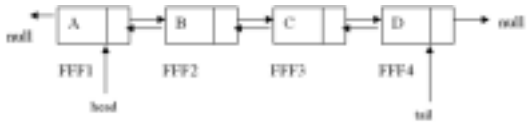
DLLNC dengan HEAD

Function untuk menghapus semua elemen

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

DLLNC 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.



DLLNC dengan HEAD dan TAIL

Inisialisasi DLLNC

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

Fungsi Inisialisasi DLLNC

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

Function untuk mengetahui kosong tidaknya DLLNC

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

DLLNC dengan HEAD dan TAIL

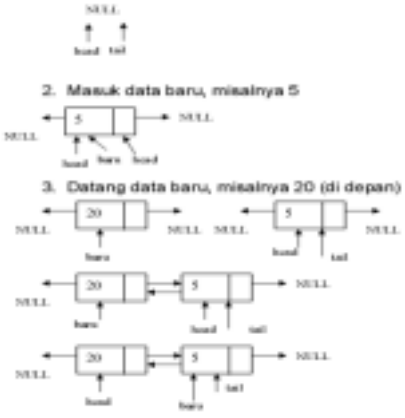
Tambah Depan

```
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;
   }
  cout<<"Data masuk\n";
}
```

DLLNC dengan HEAD dan TAIL

Ilustrasi:

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



DLLNC dengan HEAD & TAIL

Penambahan node di belakang

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

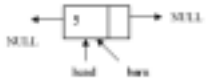
```
void insertBelakang(int databaru){
   TNode *baru;
   baru = new TNode;
   baru->data = databaru;
   baru->next = NULL;
   baru->prev = NULL;
   if(isEmpty()==1){
    head=baru;
    tail=head;
    head->next = NULL;
    head->prev = NULL;
    tail->prev = NULL;
    tail->next = NULL;
   }
   else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

DLLNC dengan HEAD & TAIL

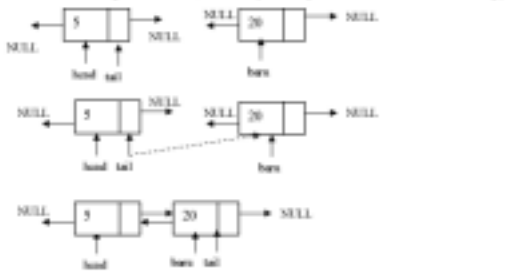
Ilustrasi:

1. List masih kosong (head=NULL)

2. Masuk data baru, misalnya 5

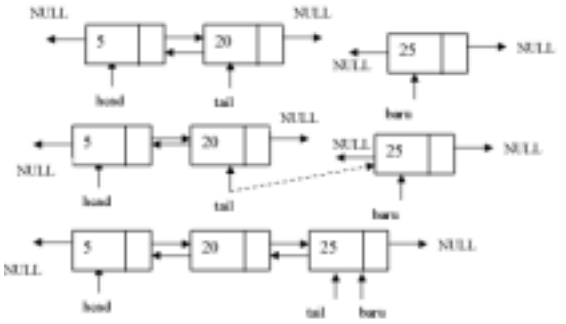


3. Datang data baru, misalnya 20 (penambahan di belakang)



DLLNC dengan HEAD & TAIL

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

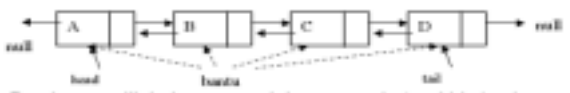


"Bagaimana dengan penambahan di tengah?"

DLLNC dengan HEAD & TAIL

- Function untuk menampilkan isi linked list

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



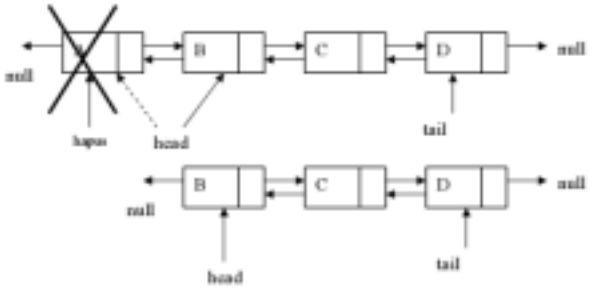
Perulangan dilakukan sampai dengan node terakhir (node `tail`)

DLLNC dengan HEAD & TAIL

Function untuk menghapus data di data terdepan

```
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";
}
```

DLLNC dengan HEAD & TAIL



DLLNC dengan HEAD & TAIL

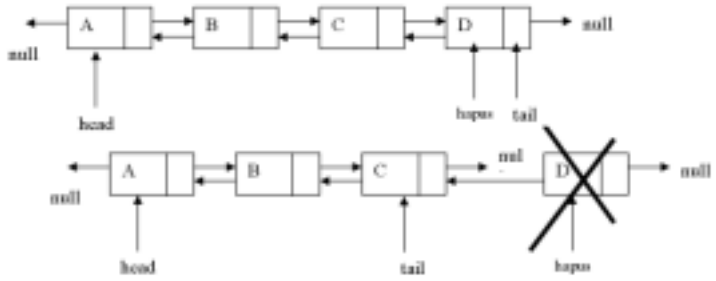
Function untuk menghapus node terbelakang

```
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";
}
```

DLLNC 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, maka pointer prev hanya perlu diset agar menunjuk ke NULL. Lalu pointer hapus didelete.

DLLNC dengan HEAD & TAIL



DLLNC 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;
}
```

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 double linked list non circular dengan menggunakan head, namun head yang ada tidak digunakan untuk menyimpan data, melainkan untuk menyimpan jumlah seluruh elemen dalam linked list.
  - Buatlah fungsi lengkap untuk tambah, hapus, lihat, dan edit!
  - Buatlah pula function untuk menampilkan data list secara terbalik!

NEXT: Double Linked List Circular