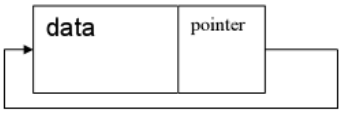
#LIST
SINGLE LINKED LIST CIRCULAR

Bahan Kuliah Struktur Data

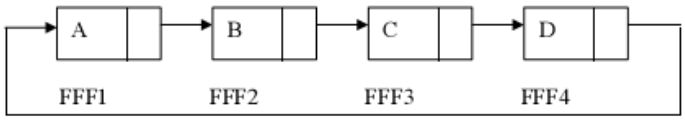
Single Linked List Circular

- SLLC adalah Single Linked List yang pointer nextnya menunjuk pada dirinya sendiri. Jika Single Linked List tersebut terdiri dari beberapa node, maka pointer next pada node terakhir akan menunjuk ke node terdepannya.
- **Pengertian:**
  - Single : artinya field pointer-nya hanya satu buah saja dan satu arah.
  - Linked List : artinya node-node tersebut saling terhubung satu sama lain.
  - Circular : artinya pointer next-nya akan menunjuk pada dirinya sendiri sehingga berputar



Menempati alamat memori tertentu

Ilustrasi SLLC



Ilustrasi SLLC

- Setiap node pada linked list mempunyai field yang berisi pointer ke node berikutnya, dan juga memiliki field yang berisi data.
- Pada akhir linked list, node terakhir akan menunjuk ke node terdepan sehingga linked list tersebut berputar.

Deklarasi dan node baru SLLC

Deklarasi node

Dibuat dari struct berikut ini:

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

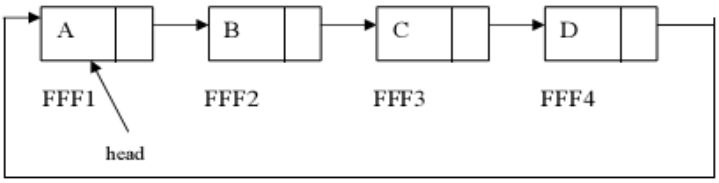
Pembentukan node baru

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

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

SLLC dengan HEAD

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



SLLC dengan HEAD

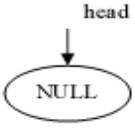
Deklarasi Pointer Penunjuk Kepala Single Linked List

- 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;
}
```



SLLC dengan HEAD

Function untuk mengetahui kosong tidaknya SLLC

```
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.

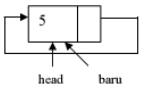
SLLC menggunakan Head

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

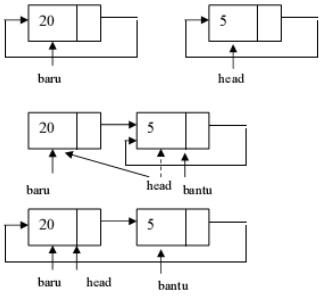
Ilustrasi:

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

2. Masuk data baru, misalnya 5



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



SLLC 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.

SLLC dengan Head

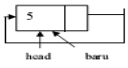
```
void insertBelakang (int databaru){
  TNode *baru,*bantu;
 baru = new TNode;
 baru->data = databaru;
 baru->next = baru;
 if(isEmpty()==1){
   head=baru;
   head->next=head;
 }
 else {
       bantu = head;
       while(bantu->next != head){
              bantu=bantu->next;
       }
       bantu->next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

SLLC dengan HEAD

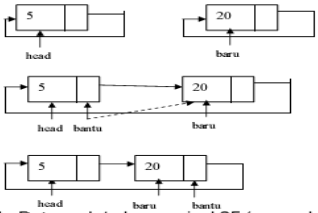
Ilustrasi:

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

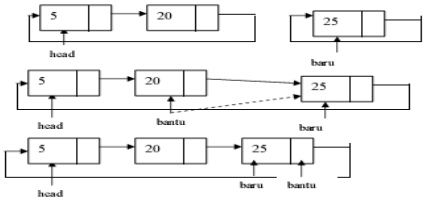
2. Masuk data baru, misalnya 5



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



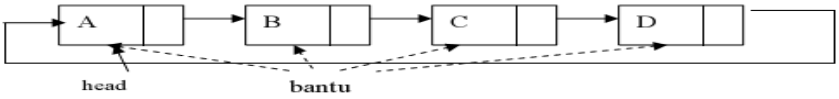
4. Datang data baru, misal 25 (penambahan di belakang)



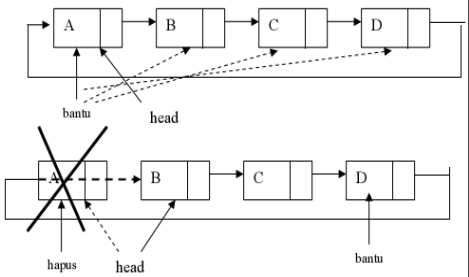
SLLC dengan HEAD

Function untuk menampilkan isi single linked list

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



SLLC dgn HEAD

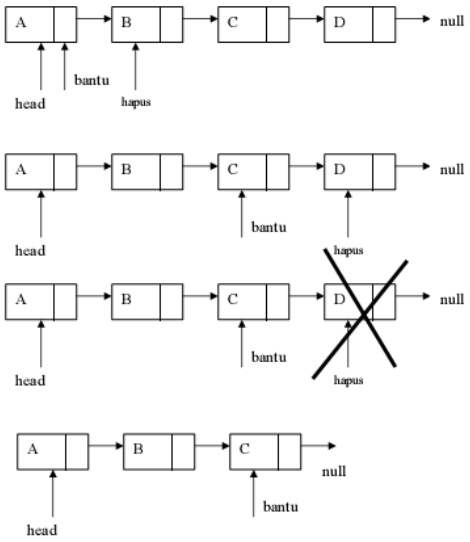



```
void hapusDepan (){
  TNode *hapus,*bantu;
  if (isEmpty()==0){
        int d;
        hapus = head;
        d = head->data;
        if(head->next != head){
                bantu = head;
                while(bantu->next!=head){
                        bantu=bantu->next;
                }
      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

SLLC dengan HEAD

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

SLLC dengan HEAD



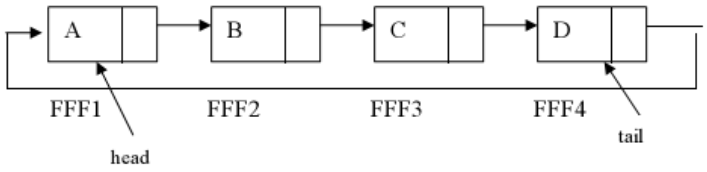
SLLC dengan HEAD

Function untuk menghapus semua elemen Linked List

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

SLLC 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.



SLLC dengan HEAD dan TAIL

Inisialisasi SLLC

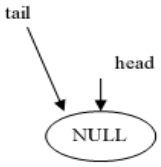
```
TNode *head, *tail;
```

Fungsi Inisialisasi SLLC

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

Function untuk mengetahui kosong tidaknya SLLC

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



SLLC dengan HEAD dan TAIL

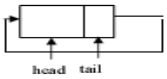
Pengkaitan node baru ke linked list di depan

Penambahan data baru di depan akan selalu menjadi head.

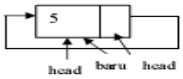
```
void insertDepan(int databaru){
  TNode *baru;
      baru = new TNode;
      baru->data = databaru;
      baru->next = baru;
      if(isEmpty()==1){
       head=baru;
       tail=baru;
       head->next=head;
       tail->next=tail;
      }
      else {
             baru->next = head;
          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

SLLC dengan HEAD dan TAIL

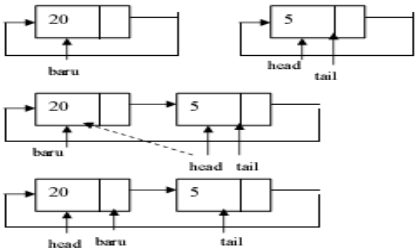
1. List masih kosong (head=tail=NULL)



2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20



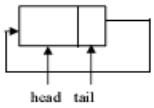
SLLC dengan HEAD & TAIL

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

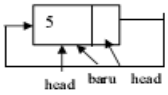
SLLC dengan HEAD & TAIL

Ilustrasi:

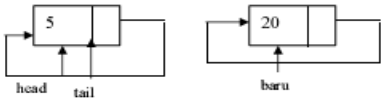
1. List masih kosong (`head=tail=NULL`)

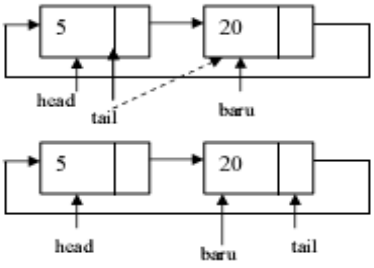


2. Masuk data baru, misalnya 5



3. Datang data baru, misalnya 20





SLLC dengan HEAD & TAIL

Function untuk menampilkan isi linked list:

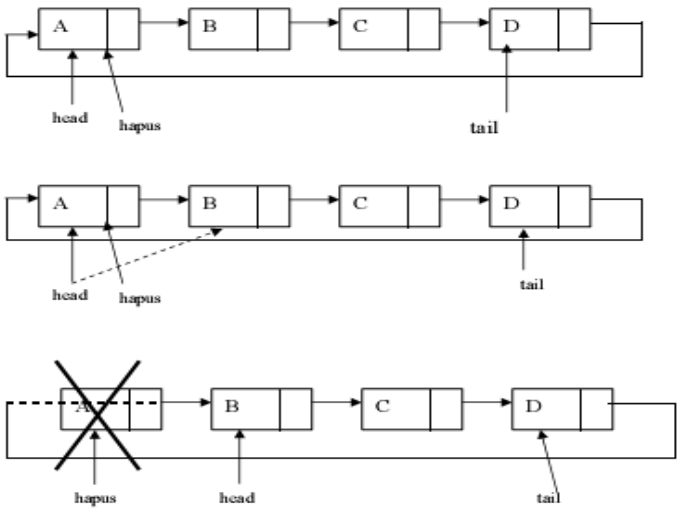
```
void tampil(){
 TNode *b;
 b = head;
 if(isEmpty()==0){
      do{
            printf("%d",b->data);
            b=b->next;
      }while(b!=tail->next);
      printf("\n");
 } else printf("Masih kosong\n");
}
```

SLLC dengan HEAD & TAIL

Function untuk menghapus data di depan

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

SLLC dengan HEAD & TAIL



SLLC 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 variabel hapus pada head, kemudian dilakukan pergeseran head ke node berikutnya sehingga data setelah head menjadi head baru, kemudian menghapus variabel hapus dengan menggunakan perintah delete.
- Jika tail masih NULL maka berarti data masih kosong!

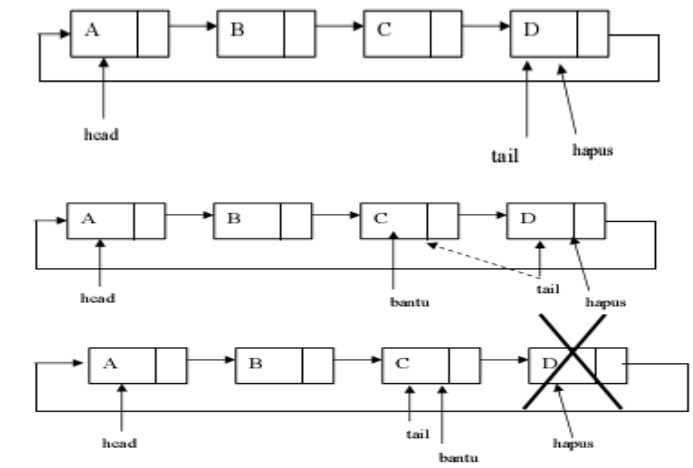
SLLC dengan HEAD & TAIL

Function untuk menghapus data di belakang:

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

SLLC dengan HEAD & TAIL

Ilustrasi:



SLLC 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 variabel hapus dengan menggunakan perintah delete.
- Jika tail masih NULL maka berarti data masih kosong!

SLLC 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;
       tail = NULL;
 }
}
```

SLLC 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 secara ber-Head maupun ber-Head dan Tail!
- Buatlah function untuk menghapus data tertentu dalam linked list!
- Buatlah penyisipan node setelah atau sebelum data tertentu.

NEXT: Double Linked List Non Circular