

# LINKED LIST

## Tujuan

Setelah mempelajari bab ini diharapkan mahasiswa akan mampu :

1. Memahami algoritma dan struktur data pada tipe linked list

## DASAR TEORI

### LINKED LIST

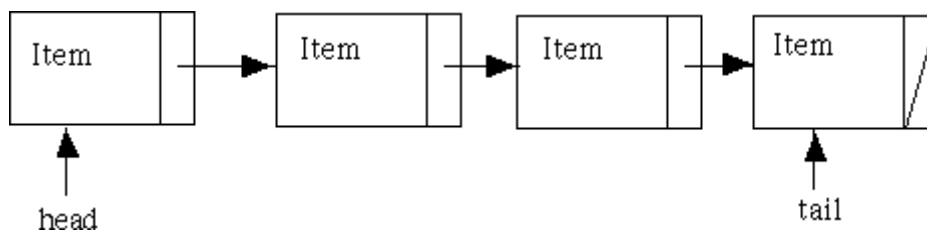
#### 1. Linked List

Linked List atau dikenal juga dengan sebutan senarai berantai adalah struktur data yang terdiri dari urutan record data dimana setiap record memiliki field yang menyimpan alamat/referensi dari record selanjutnya (dalam urutan). Elemen data yang dihubungkan dengan link pada Linked List disebut Node. Biasanya didalam suatu linked list, terdapat istilah head dan tail. Head adalah elemen yang berada pada posisi pertama dalam suatu linked list. Tail adalah elemen yang berada pada posisi terakhir dalam suatu linked list.

#### 2. Macam-Macam Linked List

##### 2.1 Single Linked List

Single Linked List merupakan suatu linked list yang hanya memiliki satu variabel pointer saja. Dimana pointer tersebut menunjuk ke node selanjutnya. Biasanya field pada tail menunjuk ke NULL. Contoh :



Contoh script :

```
struct Mahasiswa{
    char nama[25];
    int usia;
    struct Mahasiswa *next;
}*head,*tail;
```

Ada 2 Tipe Single Linked List yaitu :

### **2.1.1 Single Linked List Circular**

Single Linked List Circular 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.

Circular : artinya pointer next-nya akan menunjuk pada dirinya sendiri sehingga berputar

#### **2.1.1.1 Ilustrasi Single Linked List Circular :**

- 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. Node terakhir akan menunjuk lagi ke head.

#### **2.1.1.2 Pembuatan Single Linked List Circular**

- Deklarasi node
- Dibuat dari struct berikut ini:

```
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 haed yang bertipe pointer dari TNode yang berguna sebagai kepala linked list.

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;
```

### 2.1.1.3 Single Linked List Circular 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 melalui node pertama dalam linked list. Deklarasinya sebagai berikut:

```
TNode *head;

Fungsi Inisialisasi Single LinkedList

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

### 2.1.1.4 Function untuk mengetahui kosong tidaknya Single LinkedList

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

### 2.1.1.5 Penambahan Data

Penambahan data di depan. Penambahan node baru akan dikaitkan 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.

### 2.1.1.6 Single Linked List Menggunakan 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.

## 2.1.2 Single Linked List Non Circular

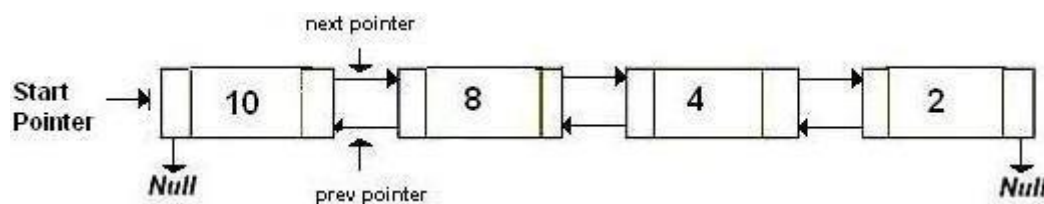
Single : artinya field pointer-nya hanya satu buah saja dan satu arah.  
Linked List : artinya node-node tersebut saling terhubung satu sama lain.

### 2.1.2.1 Ilustrasi Linked List

- 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 NULL yang akan digunakan sebagai kondisi berhenti pada saat pembacaan isi linked list.

## 2.2 Double Linked List

Double Linked List (DLL) merupakan suatu linked list yang memiliki dua variabel pointer yaitu pointer yang menunjuk ke node selanjutnya dan pointer yang menunjuk ke node sebelumnya. Setiap head dan tailnya juga menunjuk ke NULL. Contoh :



Contoh script :

```
struct Mahasiswa{
    char nama[25];
    int usia;
    struct Mahasiswa *next,*prev;
}*head,*tail;
```

### 2.2.1 Elemen double link list terdiri dari tiga bagian:

- Bagian data informasi
- Pointer next yang menunjuk ke elemen berikutnya
- Pointer prev yang menunjuk ke elemen sebelumnya

Untuk menunjuk head dari double link list, pointer prev dari elemen pertama menunjuk NULL. Sedangkan untuk menunjuk tail, pointer next dari elemen terakhir menunjuk NULL. DLL biasanya digunakan pada saat alokasi memori konvensional tidak lagi bisa diandalkan. Sedangkan bekerja dengan data yang besar tidak dapat dihindari lagi, karena tidak jarang pula, data besar tersebut memiliki hubungan yang erat. Di dalam DLL tidak hanya sekadar menampilkan setiap record-nya, melainkan dapat pula menambahkan record, menghapus beberapa record sesuai keinginan pengguna, sampai mengurutkan record. Kondisi tersebut memungkinkan dimilikinya satu rantai data yang panjang dan saling berhubungan. Pada Double Linked List, setiap node memiliki dua buah pointer ke sebelah kiri (prev) dan ke sebelah kanan (next). Gambar 1 memperlihatkan sebuah node dari Double Linked List.

Bertambah lagi komponen yang akan digunakan. Apabila dalam Single Linked List hanya memiliki head, curr dan node, maka untuk Double Linked List, ada satu penunjuk yang berfungsi sebagai akhir dari list: tail. Bagian kiri dari head akan menunjuk ke NULL. Demikian pula dengan bagian kanan dari tail. Setiap node saling terhubung dengan pointer kanan dan kiri.

## 2.2.2 Macam-Macam Double Linked List

### 2.2.2.1 Double Linked List Circular (DLLC)

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.

#### a. Bentuk Node DLLC

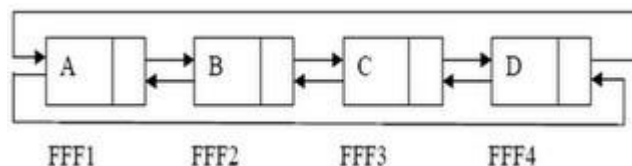
Pengertian:

Double : field pointer-nya terdiri dari dua buah dan dua arah, yaitu prev dan next.

Linked List : node-node tersebut saling terhubung satu sama lain.

Circular : pointer next dan prev-nya menunjuk ke dirinya sendiri.

Ilustrasi Double Linked List 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 dirinya sendiri.
- Jika sudah lebih dari satu node, maka pointer prev akan menunjuk ke node sebelumnya, dan pointer next akan menunjuk ke node sesudahnya.

#### b. Pembuatan Double Linked List Circular

Deklarasi node, dibuat dari struct berikut ini:

Penjelasan:

- Pembuatan struct bernama TNode yang berisi 3 field, yaitu field data bertipe integer dan field next dan prev yang bertipe pointer dari Tnode.

- Setelah pembuatan struct, buat variabel head yang bertipe pointer dari TNode yang berguna sebagai kepala linked list.

Pembuatan Node Baru:

Digunakan keyword new yang berarti mempersiapkan sebuah node baru beserta alokasi memorinya, pointer prev dan next menunjuk ke dirinya sendiri.

#### **c. Double Linked List Circular Menggunakan Head**

- Menggunakan 1 pointer head.
- Head selalu menunjuk node pertama.

Deklarasi Pointer Head:

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

#### **d. Penambahan Data**

Penambahan Data di Depan:

Penambahan node baru akan dikaitkan 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. Dibutuhkan pointer bantu yang digunakan untuk menunjuk node terakhir (head→prev) yang akan digunakan untuk mengikat list dengan node terdepan.

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, namun tidak diperlukan loop karena untuk mengetahui node terbelakang hanya perlu menunjuk pada head→prev saja. Kemudian dikaitkan dengan data baru. Untuk mengetahui data terbelakang perlu digunakan perulangan.

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

- Penelusuran dilakukan terus sampai node terakhir ditemukan menunjuk ke head lagi. Jika belum sama dengan head, 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.

**e. Penghapusan Data**

- 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 ditampung dahulu pada pointer hapus dan barulah kemudian menghapus pointer hapus dengan menggunakan perintah delete.

- Jika head masih NULL maka berarti data masih kosong.

- 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.

**f. Double Linked List Menggunakan 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.

Pengkaitan Node Baru ke Linked List di Depan:

Penambahan node baru akan selalu dikaitkan di node paling depan, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada tail/head-nya. Sedangkan jika tidak kosong, data akan ditambahkan di depan head, kemudian node baru akan berubah menjadi head.

Penambahan Node di Belakang:

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

- 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 ditampung dahulu pada variabel hapus dan barulah kemudian menghapus variabel hapus dengan menggunakan perintah delete.

- Jika tail masih NULL maka berarti data masih kosong.
- 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.

#### **2.2.2.2 Double Linked List Non Circular (DLLNC)**

Double Linked List Non 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 Non Circular pointer next dan prev nya menunjuk ke NULL. Dengan adanya 2 pointer penunjuk, next dan prev, DLLNC sangat flexible dibandingkan dengan SLLNC.

##### **a. Bentuk Node DLLC**

Pengertian:

Double : field pointer-nya terdiri dari dua buah dan dua arah, yaitu prev dan next.

Linked List : node-node tersebut saling terhubung satu sama lain.

Non Circular : pointer next dan prev-nya menunjuk ke NULL.

##### **Ilustrasi Double Linked 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.

##### **b. Pembuatan Double Linked List Non Circular**

Deklarasi node, dibuat dari struct berikut ini:

Penjelasan:

- Pembuatan struct bernama TNode yang berisi 3 field, yaitu field data bertipe integer dan field next & prev yang bertipe pointer dari Tnode.



- Setelah pembuatan struct, buat variabel head yang bertipe pointer dari TNode yang berguna sebagai kepala linked list.

Pembentukan Node Baru:

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

Untuk data pertama, pointer node baru yang prev dan next harus menunjuk ke NULL.

**c. Double Linked List Non Circular Menggunakan Head**

- Menggunakan 1 pointer head.
- Head selalu menunjuk node pertama

Deklarasi Pointer Head:

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

**d. Penambahan Data**

Penambahan Data di Depan:

Penambahan node baru akan dikaitkan 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.

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.

- 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 variabel node bantu, karena pada prinsipnya variabel node head yang menjadi tanda awal list tidak boleh berubah/berganti posisi.

- Penelusuran dilakukan terus sampai node terakhir ditemukan menunjuk NULL. Jika belum 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.
- 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 ditampung dahulu pada pointer hapus dan barulah kemudian menghapus pointer hapus dengan menggunakan perintah delete. Namun sebelumnya pointer head harus menunjuk terlebih dahulu ke node selanjutnya.
- Jika head masih NULL maka berarti data masih kosong.
- 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.

#### **e. Double Linked List Menggunakan 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.

Pengkaitan Node Baru ke Linked List di Depan:

Penambahan node baru akan selalu dikaitkan di node paling depan, namun pada saat pertama kali (data masih kosong), maka penambahan data dilakukan pada tail/head nya. Sedangkan jika tidak kosong, data akan ditambahkan didepan head, kemudian node baru akan berubah menjadi head.

Penambahan Node di Belakang:

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

- 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 ditampung dahulu pada pointer hapus dan barulah kemudian menghapus pointer hapus dengan menggunakan perintah delete. Namun

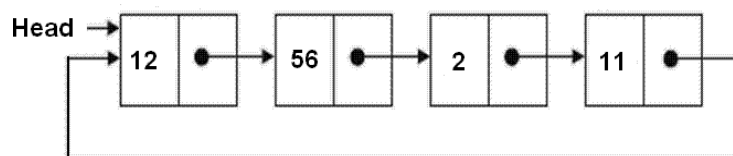
sebelumnya pointer head harus ditunjuk lebih dahulu ke node sesudahnya agar tetap menunjuk ke node terdepan.

- Jika tail masih NULL maka berarti data masih kosong.
- 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 di delete.

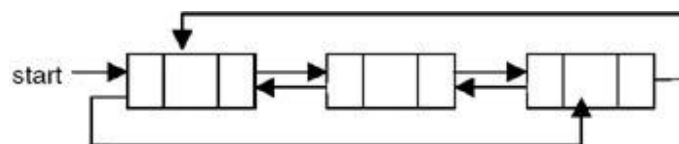
### 2.3 Circular Linked List

Circular Linked List merupakan suatu linked list dimana tail (node terakhir) menunjuk ke head (node pertama). Jadi tidak ada pointer yang menunjuk NULL. Ada 2 jenis Circular Linked List, yaitu :

#### 2.3.1 Circular Single Linked List

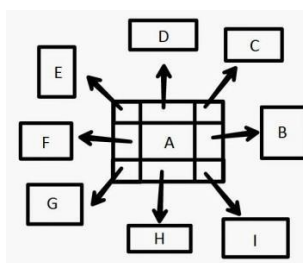


#### 2.3.2 Circular Double Linked List



### 2.4 Multiplid Linked List

Multiple Linked List merupakan suatu linked list yang memiliki lebih dar 2 buat variabel pointer. contoh :



## Latihan 1 NO 1

### Pemrograman Java

**Nama Program** : package P9NO1  
**Bahasa Pemrograman** : Java  
**Compiler** : NetBeans IDE 8.2  
**Script program** :

#### Class Node

```
1  package p9no1;
2  //@Dwithafr @Ikadm_
3  public class Node {
4      int data;
5      Node next;
6
7      public Node(int data) {
8          this.data = data;
9      }
10
11     public void tampil(){
12         System.out.print("{ "+data+" }");
13     }
14 }
```

#### Class LinkedList

```
1  package p9no1;
2  //@Dwithafr @Ikadm_
3  public class LinkedList {
4      Node first;
5
6      public LinkedList() {
7          first = null;
8      }
9
10     public boolean isEmpty(){
11         return (first==null);
12     }
13
14
15     public void addFirst(int data){
16         Node node = new Node(data);
17         node.next = first;
18         first = node;
19     }
20
21     // menambah data dari simpul terakhir
22     public void addLast(int data){
23         Node node, help;
24         node = new Node(data);
25         node.next = null;
```

```

26
27         if(isEmpty()){
28             first = node;
29             first.next = null;
30         }else{
31             help = first;
32             while(help.next!=null){
33                 help=help.next;
34             }
35             help.next=node;
36         }
37     }
38
39     // Menghapus data dari simpul pertama
40     public Node deleteFirst(){
41         if(!isEmpty()){
42             Node temp = first;
43             first = first.next;
44             return temp;
45         }else{
46             return null;
47         }
48     }
49
50     // Menghapus data dari simpul terakhir
51     public Node deleteLast(){
52         if(!isEmpty()){
53             Node temp, current;
54             current=first;
55             while(current.next.next != null){
56                 current=current.next;
57             }
58             temp=current.next;
59             current.next=null;
60             return temp;
61         }else{
62             Node temp = first;
63             first = null;
64             return temp;
65         }
66     }
67
68     // Menampilkan isi linked list
69     public void tampilkan(){
70         Node current = first;
71         if(current == null){
72             System.out.println("Kosong!");
73         }else{

```

```

74         while(current != null){
75             current.tampil();
76             current = current.next;
77         }
78         System.out.println();
79     }
80 }
81 }

```

#### Class LinkedListApp

```

1  package p9no1;
2  //@Dwithafr @Ikadm_
3  public class LinkedListApp {
4      public static void main(String[] args) {
5          LinkedList link = new LinkedList();
6          link.addFirst(1);
7          link.addFirst(2);
8          link.addLast(3);
9          link.addLast(4);
10         link.tampilkan();
11         link.deleteLast();
12         link.tampilkan();
13     }
14 }

```

#### Output Program :

```

Output - P9NO1 (run)
run:
{2}{1}{3}{4}
{2}{1}{3}
BUILD SUCCESSFUL (total time: 0 seconds)

```

#### Penjelasan Program :

Pada program ini praktikkan membuat 3 class public dengan nama class Node, class LinkedList dan class LinkedListApp.

Node.java	
1	Package dari class Node pada package p9no1
2	Komentar author
3-6	Pembuatan class public bernama Node yang berisi 2 field/variabel, yaitu data yang bertipe int dan next yang bertipe class Node
7-10	Deklarasi constructor class Node; dengan parameter (int data) inialisasi variable this.data sama dengan nilai dari data.
11-14	Method tampil yang digunakan untuk menampilkan data.

LinkedList.java	
1	Package dari class LinkedList pada package p9no1
2	Komentar author
3	Pembuatan class bernama LinkedList yang berisi 1 field/variabel, yaitu first yang bertipe class Node.
6-8	Deklarasi constructor class LinkedList; inialisasi variabel first dengan nilai null.
10-13	Method isEmpty, mengembalikan nilai first sama dengan null.
15-19	Method untuk menambah data di depan; parameter variabel data dengan tipe integer.
22-37	Method untuk menambah data di belakang (akhir); parameter variabel data dengan tipe integer.
40-48	Method untuk menghapus data pertama.
51-64	Method untuk menghapus data terakhir.
69-81	Method untuk menampilkan data.

LinkedListApp.java	
1	Package dari class LinkedListApp pada package p9no1
2	Komentar author
3	Pembuatan class bernama LinkedListApp yang berisi main method.
4-14	Deklarasi link sama dengan LinkedList baru; Deklarasi dengan pemanggilan method addFirst(1) berarti menambahkan angka 1 pada pertama. Deklarasi dengan pemanggilan method addFirst(2) berarti menambahkan angka 2 pada sebelum data pertama atau data 1. Deklarasi dengan pemanggilan method addLast(3) berarti menambahkan angka 3 pada terakhir. Deklarasi dengan pemanggilan method addLast(4) berarti menambahkan angka 4 pada setelah data terakhir atau data 3. Pemanggilan method tampilkan, untuk menampilkan data. Pemanggilan method deleteLast, untuk menghapus data terakhir. Pemanggilan method tampilkan, untuk menampilkan data.

Maka output dari latihan 1 ini dapat disimulasikan sebagai berikut :

Empty	Sehingga pada saat pemanggilan method tampilkan yang pertama menampilkan keluaran : {2},{1},{3},{4}  Namun pada saat terjadi pemanggilan method deleteLast, maka data terakhir dihapus, sehingga pada saat pemanggilan method tampilkan yang kedua menampilkan keluaran : {2},{1},{3}
addFirst(1)	
addFirst(2)	
addLast(3)	
addLast(4)	
DeleteLast	

## Latihan 1 NO 2

Pemrograman Java

**Nama Program** : package P9NO1

**Bahasa Pemrograman** : Java

**Compiler** : NetBeans IDE 8.2

**Script program** :

**Class Node2**

```
1 package p9no1;
2 // @Dwithafr @Ikadm_
3 public class Node2 {
4     int data;
5     Node2 next;
6     Node2 prev;
7     public Node2(int data) {
8         this.data = data;
9     }
10    public void tampil() {
11        System.out.print(+data+"->");
12    }
13 }
```

**Class LL**

```
1 package p9no1;
2 // @Dwithafr @Ikadm_
3 public class LL {
4     Node2 first;
5     Node2 last;
6     public LL() {
7         first = null;
8     }
9     public boolean apaKosong() {
10        return (first == null);
11    }
12    public void dataAwal(int data) {
13        Node2 node = new Node2(data);
14        node.next = first;
15        first = node;
16    }
17    public void dataAkhir(int data) {
18        Node2 node, help;
19        node = new Node2(data);
20        node.next = null;
21
22        if (apaKosong()) {
23            first = node;
24            first.next = null;
25        } else {
26            help = first;
```



```

27         while (help.next!=null) {
28             help=help.next;
29         }
30         help.next=node;
31     }
32 }
33 public Node2 hapusAwal() {
34     if (!apaKosong()) {
35         Node2 temp = first;
36         first = first.next;
37         return temp;
38     } else {
39         return null;
40     }
41 }
42 public Node2 hapusAkhir() {
43     if (!apaKosong()) {
44         Node2 temp, current;
45         current=first;
46         while (current.next.next != null) {
47             current=current.next;
48         }
49         temp=current.next;
50         current.next=null;
51         return temp;
52     } else {
53         Node2 temp = first;
54         first = null;
55         return temp;
56     }
57 }

58 public void hapusTerserah(int key) {
59     Node2 temp = first;
60     if (!apaKosong()) {
61         while (temp!= null) {
62             if (temp.next.data==key) {
63                 temp.next = temp.next.next;
64                 break;
65             }
66             else if ((temp.data==key) && (temp==first)) {
67                 this.hapusAwal();
68                 break;
69             }
70             temp=temp.next;
71         }
72     }
73 }
74 public void show() {
75     Node2 current = first;
76     if (current==null) {

```

```

77         System.out.println("Kosong");
78     }else{
79         while(current != null){
80             current.tampil();
81             current = current.next;
82         }
83         System.out.println();
84     }
85 }
86 }

```

#### Class LLApp

```

1  package p9no1;
2  //@Dwithafr @Ikadm_
3  public class LLApp {
4      public static void main(String[] args){
5          LL link = new LL();
6          System.out.println("1 : LL ASAL");
7          link.dataAwal(11);
8          link.dataAwal(2);
9          link.dataAwal(30);
10         link.dataAwal(14);
11         link.dataAwal(5);
12         link.dataAwal(16);
13         link.show();
14         System.out.println("1 : LL SETELAH SISIP DI AKHIR");
15         link.dataAkhir(56);
16         link.dataAkhir(16);
17         link.show();
18         System.out.println("1 : LL SETELAH DIHAPUS DI DEPAN");
19         link.hapusAwal();
20         link.show();
21         System.out.println("LL SETELAH 30 DIHAPUS");
22         link.hapusTerseher(30);
23         link.show();
24     }
25 }

```

#### Output Program :

```

Output - P9NO1 (run)
run:
1 : LL ASAL
16->5->14->30->2->11->
1 : LL SETELAH SISIP DI AKHIR
16->5->14->30->2->11->56->16->
1 : LL SETELAH DIHAPUS DI DEPAN
5->14->30->2->11->56->16->
LL SETELAH 30 DIHAPUS
5->14->2->11->56->16->
BUILD SUCCESSFUL (total time: 0 seconds)

```

## Penjelasan Program :

Pada program ini praktikkan membuat 3 class public dengan nama class Node2, class LL dan class LLApp.

Node2.java	
1	Package dari class Node pada package p9no1
2	Komentar author
3-6	Pembuatan class public bernama Node2 yang berisi 3 field/variabel, yaitu data yang bertipe int dan next, prev yang bertipe class Node
7-9	Deklarasi constructor class Node; dengan parameter (int data) inialisasi variable this.data sama dengan nilai dari data.
10-13	Method tampil yang digunakan untuk menampilkan data.

LL.java	
1	Package dari class LinkedList pada package p9no1
2	Komentar author
3	Pembuatan class bernama LL yang berisi 2 field/variabel, yaitu first, last yang bertipe class Node.
6-8	Deklarasi constructor class LL; inialisasi variabel first dengan nilai null.
9-11	Method apaKosong, mengembalikan nilai first sama dengan null.
12-16	Method untuk menambah data di depan; parameter variabel data dengan tipe integer.
17-32	Method untuk menambah data di belakang (akhir); parameter variabel data dengan tipe integer.
33-41	Method untuk menghapus data pertama.
42-57	Method untuk menghapus data terakhir.
58-73	Method untuk menghapus data sesuai keinginan.
74-86	Method untuk menampilkan data.

LLApp.java	
1	Package dari class LLApp pada package p9no1
2	Komentar author
3	Pembuatan class bernama LLApp yang berisi main method.
5-2	Deklarasi link sama dengan LL baru; Cetak 1 : LL ASAL Deklarasi dengan pemanggilan method dataAwal(11) berarti menambahkan angka 11 pada pertama. Deklarasi dengan pemanggilan method dataAwal(2) berarti menambahkan angka 2 pada sebelum data pertama atau data 11. Deklarasi dengan pemanggilan method dataAwal(30) berarti menambahkan angka 30 pada sebelum data pertama atau data 2. Deklarasi dengan pemanggilan method dataAwal(14) berarti menambahkan angka 14 pada sebelum data pertama atau data 30.

	<p>Deklarasi dengan pemanggilan method dataAwal(5) berarti menambahkan angka 5 pada sebelum data pertama atau data 14.</p> <p>Deklarasi dengan pemanggilan method dataAwal(16) berarti menambahkan angka 16 pada sebelum data pertama atau data 5.</p> <p>Pemanggilan method show untuk menampilkan data.</p> <p>Cetak 1 : LL SETELAH SISIP DI AKHIR</p> <p>Deklarasi dengan pemanggilan method dataAkhir(56) berarti menambahkan angka 56 pada terakhir.</p> <p>Deklarasi dengan pemanggilan method dataAkhir(16) berarti menambahkan angka 16 pada setelah data terakhir atau data 56.</p> <p>Pemanggilan method show, untuk menampilkan data.</p> <p>Cetak 1 : LL SETELAH DIHAPUS DI DEPAN</p> <p>Pemanggilan method hapusAwal, untuk menghapus data pertama.</p> <p>Pemanggilan method show, untuk menampilkan data.</p> <p>Cetak LL SETELAH 30 DIHAPUS</p> <p>Pemanggilan method hapusTerserah(30), untuk menghapus data dengan nilai 30.</p> <p>Pemanggilan method show, untuk menampilkan data.</p>
--	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Maka output dari latihan 1 ini dapat disimulasikan sebagai berikut :

Empty						
dataAwal(11)	11					
dataAwal(2)	2 11					
dataAwal(30)	30 2 11					
dataAwal(14)	14 30 2 11					
dataAwal(5)	5 14 30 2 11					
dataAwal(16)	16 5 14 30 2 11					
dataAkhir(56)	16 5 14 30 2 11 56					
hapusAwal()	5 14 30 2 11 56					
hapusTerserah(30)	5 14 2 11 56					

Pada saat pemanggilan method show yang pertama menampilkan keluaran :  
16 → 5 → 14 → 30 → 2 → 11 → NULL

Namun pada saat terjadi penambahan data pada akhir maka elemen nya akan berbeda lagi, sehingga akan menampilkan keluaran :  
16 → 5 → 14 → 30 → 2 → 11 → 56 → 16 → NULL

Keluaran ketiga adalah :  
5 → 14 → 30 → 2 → 11 → 56 → 16 → NULL karena data awal dihapus

Keluaran keempat adalah :  
5 → 14 → 2 → 11 → 56 → 16 → NULL karena data 30 dihapus.

## Latihan 2

### Pemrograman Java

**Nama Program** : package P9NO2  
**Bahasa Pemrograman** : Java  
**Compiler** : NetBeans IDE 8.2  
**Script program** :



#### Class Node

```
1 package p9no2;
2 // @Dwithafr @Ikadm_
3 public class Node {
4     int data;
5     Node next;
6     Node prev;
7
8     public Node(int data) {
9         this.data = data;
10    }
11
12    public void tampil() {
13        System.out.print("(" + data + " ");
14    }
15 }
```

#### Class DoubleLinkedList

```
1 package p9no2;
2 // @Dwithafr @Ikadm_
3 public class DoubleLinkedList {
4     Node first;
5     Node last;
6
7     //kontruktor
8     //set nilai awal adalah null
9     public DoubleLinkedList() {
10        first = null;
11        last = null;
12    }
13
14    //mengecek apakah linked list kosong atau tidak
15    public boolean isEmpty() {
16        return (first == null);
17    }
18
19    //method untuk menginsert data dari pertama
20    public void insertFirst(int data) {
21        Node node = new Node(data);
22        if (isEmpty()) {
23            last = node;
24        } else {
25            first.prev = node;
```

```

26     }
27
28     node.next = first;
29     first = node;
30 }
31
32 //method untuk menginsert data dari terakhir
33 [- public void insertLast(int data) {
34     Node node = new Node (data);
35     if( isEmpty() )
36         first = node;
37     else{
38         last.next = node;
39         node.prev = last;
40     }
41     last = node;
42 }
43
44 //method untuk menginsert data pertama
45 [- public Node deleteFirst(){
46     Node temp = first;
47     if(first.next == null)
48         last = null;
49     else
50         first.next.prev = null;
51      first = first.next;
52     return temp;
53 }
54
55 //method untuk menghapus data terakhir
56 [- public Node deleteLast(){
57     Node temp = last;
58
59     if(first.next == null)
60         first = null;
61     else
62         last.prev.next = null;
63      last = last.prev;
64     return temp;
65 }
66
67 //method untuk menginsert data di tengah
68 [- public boolean insertAfter(int key, int data){
69     Node current = first;
70     while(current.data !=key){
71         current = current.next;
72         if(current == null)
73             return false;
74     }
75     Node node = new Node(data);

```

```

76         if(current == last){
77             node.next = null;
78             last = node;
79         }else{
80             node.next = current.next;
81
82             current.next.prev = node;
83         }
84         node.prev = current;
85         current.next = node;
86         return true;
87     }
88
89     //method untuk menghapus data yang dipilih
90     public Node deleteKey(int key){
91         Node current = first;
92         while(current.data !=key){
93             current = current.next;
94             if(current == null)
95                 return null;
96         }
97         if(current == first)
98             first = current.next;
99         else
100             current.prev.next = current.next;
101         if(current == last)
102             last = current.next.prev = current.prev;
103         return current;
104     }
105
106     //menampilkan data dari pertama - terakhir
107     public void displayForward(){
108         System.out.print("List (first-->last): ");
109         Node current = first;
110         while(current != null){
111             current.tampil();
112             current = current.next;
113         }
114         System.out.println("");
115     }
116
117     //menampilkan data dari terakhir - pertama
118     public void displayBackward(){
119         System.out.print("List (last-->first): ");
120         Node current = last;
121         while(current != null){
122             current.tampil();
123             current = current.prev;
124         }
125         System.out.println("");
126     }
127 }

```

### Class DoubleLinkedListApp

```
1 package p9no2;
2 // @Dwithafr @Ikadm_
3 public class DoubleLinkedListApp {
4     public static void main(String[] args) {
5         DoubleLinkedList link = new DoubleLinkedList();
6         link.insertFirst(1);
7         link.insertFirst(2);
8         link.insertLast(3);
9         link.insertLast(4);
10        link.displayForward();
11        link.displayBackward();
12        link.insertAfter(3,10);
13        link.displayForward();
14        link.displayBackward();
15        link.deleteFirst();
16    }
17 }
```

### Output Program

```
Output - P9NO2 (run)
run:
List (first-->last): (2) (1) (3) (4)
List (last-->first): (4) (3) (1) (2)
List (first-->last): (2) (1) (3) (10) (4)
List (last-->first): (4) (10) (3) (1) (2)
BUILD SUCCESSFUL (total time: 0 seconds)
```

### Penjelasan Program

Pada program ini praktikkan membuat 3 class public dengan nama class Node, class DoubleLinkedList dan class DoubleLinkedListApp.

Node.java	
1	Package dari class Node pada package p9no2
2	Komentar author
3-6	Pembuatan class public bernama Node yang berisi 3 field/variabel, yaitu data yang bertipe int dan prev, next yang bertipe class Node
8-10	Deklarasi constructor class Node; dengan parameter (int data) inialisasi variable this.data sama dengan nilai dari data.
12-15	Method tampil yang digunakan untuk menampilkan data.



<b>DoubleLinkedList.java</b>	
1	Package dari class DoubleLinkedList pada package p9no2
2	Komentar author
3-5	Pembuatan class bernama DoubleLinkedList yang berisi 2 field/variabel, yaitu first, last yang bertipe class Node.
9-12	Deklarasi constructor class DoubleLinkedList; inisialisasi variabel first, last sama dengan nilai null.
15-16	Method isEmpty, mengembalikan nilai first sama dengan null.
20-30	Method untuk menambah data di depan; parameter variabel data dengan tipe integer.
33-42	Method untuk menambah data di belakang (akhir); parameter variabel data dengan tipe integer.
45-53	Method untuk menghapus data pertama.
56-64	Method untuk menghapus data terakhir.
67-87	Method untuk menambah data setelah data tertentu; parameter variabel key, data dengan tipe data integer.
90-104	Method untuk menghapus data key; parameter variabel key dengan tipe data integer.
106-115	Method untuk menampilkan data dari awal hingga akhir.
118-127	Method untuk menampilkan data dari akhir hingga awal.

<b>DoubleLinkedListApp.java</b>	
1	Package dari class DoubleLinkedListApp pada package p9no2
2	Komentar author
3	Pembuatan class bernama DoubleLinkedListApp yang berisi main method.
4-17	<p>Deklarasi link sama dengan DoubleLinkedList baru;</p> <p>Deklarasi dengan pemanggilan method insertFirst(1) berarti menambahkan angka 1 pada pertama.</p> <p>Deklarasi dengan pemanggilan method insertFirst(2) berarti menambahkan angka 2 pada sebelum data pertama atau data 1.</p> <p>Deklarasi dengan pemanggilan method insertLast(3) berarti menambahkan angka 3 pada terakhir.</p> <p>Deklarasi dengan pemanggilan method insertLast(4) berarti menambahkan angka 4 pada setelah data terakhir atau data 3.</p> <p>Pemanggilan method displayForward, untuk menampilkan data dari awal hingga akhir.</p> <p>Pemanggilan method displayBackward, untuk menampilkan data dari akhir hingga awal.</p> <p>Deklarasi dengan pemanggilan method insertAfter(3,10) berarti menambahkan angka 10 pada setelah data 3.</p> <p>Pemanggilan method displayForward, untuk menampilkan data dari awal hingga akhir.</p> <p>Pemanggilan method displayBackward, untuk menampilkan data dari akhir hingga awal.</p> <p>Pemanggilan method deleteFirst, untuk menghapus data awal.</p>

Maka output dari latihan 2 ini dapat disimulasikan sebagai berikut :

Empty	<p>Sehingga pada saat pemanggilan method displayForward yang pertama menampilkan keluaran :  {2},{1},{3},{4}</p> <p>Namun pada saat terjadi pemanggilan</p>
insertFirst(1)	
1	
insertFirst(2)	

2	1			
insertLast(3)				
2	1		3	
insertLast(4)				
2	1	3	4	
insertAfter(3,10)				
2	1	3	10	4
deleteFirst				
	1	3	10	4

method displayBackward, maka data ditampilkan dari akhir, sehingga pada saat pemanggilan method tersebut menampilkan keluaran :  
{4},{3},{1},{2}

Kemudian saat penambahan nilai 10 setelah data yang bernilai 3 dengan pemanggilan method addAfter maka susunannya menjadi  
2 → 1 → 3 → 10 → 4

Pada saat pemanggilan method displayForward yang kedua menampilkan keluaran :  
{2},{1},{3},{10},{4}

Sedangkan pada saat terjadi pemanggilan method displayBackward, maka data ditampilkan dari akhir, sehingga pada saat pemanggilan method tersebut menampilkan keluaran :  
{4},{10},{3},{1},{2}

Kemudian data yang pertama dihapus menggunakan method delete first sehingga data {2} dihapus.

### Latihan 3

Pemrograman Java

Nama Program : package P9NO3

Bahasa Pemrograman : Java

Compiler : NetBeans IDE 8.2

Script program :

Class CircularLinkedList

```
1  package p9no3;
2  //@Dwithafr @Ikadm_
3  public class CircularLinkedList {
4  private Node start;
5  private int count;
6  public void append(int x) {
7      count++;
8      Node temp=new Node(x);
9      if(start==null){
10         start=temp;
11     }else{
12         Node tp=start;
13         while(tp.link!=start){
14             tp=tp.link;
15         }
16         tp.link=temp;
17     }
18     temp.link=start;
19 }
20 public void addBeg(int x){
21     count++;
22     Node temp=new Node(x);
23     if(start==null){
24         temp.link=temp;
25     }else{
26         Node tp=start;
27         while(tp.link!=start){
28             tp=tp.link;
29         }
30         tp.link=temp;
31         temp.link=start;
32     }
33     start=temp;
34 }
35 public void addAt(int pos,int x){
36     Node temp,tp;
37     temp=new Node(x);
38     tp=start;
39     for(int i=0;i<pos;i++){
40         if(tp.link==start)
```

```

41     break;
42     tp=tp.link;
43 }
44 temp.link=tp.link;
45 tp.link=temp;
46 count++;
47 }
48 public void displayList() {
49     if(start==null)
50         System.out.println("List is empty..");
51     else{
52         Node temp=start;
53         System.out.print("->");
54         while(temp.link!=start) {
55             System.out.println(" "+temp.data);
56             temp=temp.link;
57         }
58         System.out.println(temp.data+" ->");
59     }
60 }
61 public void deleteAt(int position) {
62     Node current=start;
63     Node previous=start;
64     for(int i=0;i<position;i++){
65         if(current.link==start)
66             break;
67         previous=current;
68         current=current.link;
69     }
70     System.out.print(current.data);
71     if(position==0)
72         deleteFirst();
73     else
74         previous.link=current.link;
75     count--;
76 }
77 public void deleteFirst() {
78     Node temp=start;
79     while(temp.link!=start) {
80         temp=temp.link;
81     }
82     temp.link=start.link;
83     start=start.link;
84     count--;
85 }
86 public int getCount() {
87     return count;
88 }
89 private static class Node{
90     int data;
91     Node link;

```

```

92 public Node(int data){
93     this.data=data;
94 }
95 public Node(int data,Node link){
96     this.data=data;
97     this.link=link;
98 }
99 }
100 }

```

#### Class CLLUserClass

```

1 package p9no3;
2 // @Dwithafr @Ikadm_
3 public class CLLUserClass {
4     public static void main (String args []){
5         CircularLinkedList ccl=new CircularLinkedList();
6         ccl.addBeg (1);
7         ccl.append (2);
8         ccl.append (3);
9         ccl.append (4);
10        ccl.addAt (1,0);
11        ccl.append (5);
12        ccl.append (12);
13        ccl.displayList();
14        ccl.deleteAt (1); //index starts from zero
15        System.out.println ("After deletion .. .");
16        ccl.displayList();
17    }
18 }

```

#### Output Program

```

Output - P9NO3 (run)

run:
-> 1
2
0
3
4
5
12 ->
2After deletion .. .
-> 1
0
3
4
5
12 ->
BUILD SUCCESSFUL (total time: 0 seconds)

```

## **Penjelasan Program :**

Pada program diatas menggunakan dua kelas yang terdapat dala satu package, yang bernama P9\_3. Dua class tersebut antara lain adalah CircularLinkedList dan CLLUserClass.

- CircularLinkedList.java

Class ini berisi method-method yang akan dijalankan melalui class CLLUserClass. Method yang terdapat di class ini diantaranya adalah append(), addBeg(), addAt(), displayList(), deleteAt(), deleteFirst(), dan getCount(). Pada method append() digunakan untuk menambahkan data(record) pada circular linked list. Method addBeg() digunakan untuk menambahkan data pada bagian mulainya data(begin). Method addAt() digunakan untuk menambahkan data pada posisi tertentu yang terpilih, dalam tanda kurung dimasukkan awalnya berisi data yang menjadi acuan kemudian dilanjutkan dengan nilai data yang akan dimasukkan, misal addAt(1, 5). Method displayList() digunakan untuk menampilkan isi yang ada pada circular linked list, jika linked list kosong atau tidak ada isinya akan muncul kalimat "List is empty..", dan jika tidak kosong maka data yang ada pada linked list akan ditampilkan. Method deleteAt() digunakan untuk menghapus data pada posisi yang dipilih, sedangkan method deleteFirst() digunakan untuk menghapus data paling awal tetapi bukan data pada begin. Dan yang terakhir method getCount() digunakan menghitung banyaknya data pada circular liked list.

- CLLUserClass.java

Class ini digunakan untuk memuat method untuk memanggil method-method lainnya yang terdapat dalam class CircularLinkedList.

## KESIMPULAN

Dari praktikum kali ini, kami dapat menyimpulkan bahwa :

- Node adalah tempat penyimpanan data pada Linked List dimana terdiri dari dua bagian/field yakni field data dan pointer. Pointer (link) adalah field untuk menyimpan alamat tertentu.
- ADT SLL adalah struktur data yang dibangun dari satu atau lebih node dimana pointer-nya hanya satu buah dan satu arah, yaitu menunjuk ke node sesudahnya dan node terakhir akan menunjuk ke null. ADT SLL berguna untuk mengolah suatu kumpulan data yg sifatnya dinamik serta pengaksesannya cukup dari satu arah saja.
- Perbedaan ADT SLL dan DLL adalah : SLL hanya bergerak kesatu arah saja, tidak menunjuk pointer didepannya sehingga tidak dapat kembali ke pointer - pointer sebelumnya, Sedangkan DLL dapat bergerak kedua arah karena memiliki pointer next yang menunjuk ke elemen berikutnya, dan Pointer prev yang menunjuk ke elemen sebelumnya.
- ADT DLL berguna untuk mengolah data yang bersifat dinamik, dan membutuhkan pengaksesan linkedlist dari dua arah, dari kanan-kiri, maju- mundur, atau atas-bawah, dsb.
- ADT Circular Linked List berguna untuk menyimpan banyak data yang bersifat dinamis, misal untuk menyisipkan dan menghapus data. Untuk data yang membutuhkan pengaksesan dari satu arah saja digunakan CSLL, sedangkan jika membutuhkan pengaksesan dari dua arah, kita dapat menggunakan CDLL.

## **DAFTAR RUJUKAN**

- Annisa Puspa Kirana, S.Kom, M.Kom. 2016. Modul Praktikum Algoritma dan Struktur Data. Malang : Universitas Negeri Malang.
- Susanti. 2013. Single LinkList, <http://susanti.ilearning.me/2013/10/03/single-link-list/>, diakses pada tanggal 05 April 2017, pukul 20:21.
- Novi, Sucianti. 2014. LinkedList, [http://suciantinovi.blogspot.co.id/2014/03/linked-list-i\\_14.html](http://suciantinovi.blogspot.co.id/2014/03/linked-list-i_14.html), diakses pada tanggal 05 April 2017, pukul 20:43.
- Mike. 2009. Double Linked List, <http://mikeprastiwi.blogspot.co.id/2009/05/double-linked-list.html>, diakses pada tanggal 05 April 2017, pukul 21:03.
- Brawly. 2014. Double Linked List, <http://brawlyvonfabre.blogspot.co.id/p/double-linked-list.html>, diakses pada tanggal 05 April 2017, pukul 21:10.
- Al-Malik, Dani. 2013. Double Linked List, <http://danielmalik275.blogspot.co.id/2013/04/double-linked-list.html>, diakses pada tanggal 05 April 2017, pukul 23:00.