LAPORAN HASIL PRAKTIKUM
ALGORITMA STRUKTUR DATA
JOBSHEET 12

NAMA : JIRO AMMAR WAFI

NIM : 244107020190

KELAS : 1-E

PROGRAM STUDI TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLINEMA

2025

|| JOBSHEET XII - Double Linked-List

1. Percobaan 1

A. Class Mahasiswa14

1. Class ini dibuat untuk mengisi data yang akan disimpan ke dalam node pada linked-list, serta method untuk mencetak data.

```java
public class Mahasiswa14 {
    public String nim, nama, kelas;
    public double ipk;

    public Mahasiswa14 (String nim, String nama, String kelas, double ipk) {
        this.nim = nim;
        this.nama = nama;
        this.kelas = kelas;
        this.ipk = ipk;
    }

    public void tampil() {
        System.out.println("NIM: " + nim + ", Nama: " + nama + ", Kelas: " + kelas + ", IPK: " + ipk);
    }
}
```

B. Class Node14

1. Class ini digunakan untuk menginisialisasi node yang berisi data berupa object dari mahasiswa14 dan pointer menuju node selanjutnya yaitu next dan node sebelumnya yaitu prev. Serta konstruktor berparameter untuk membuat node..

```java
public class Node14 {
    Mahasiswa14 data;
    Node14 prev, next;

    public Node14 (Mahasiswa14 data) {
        this.data = data;
        prev = null;
        next = null;
    }
}
```

C. Class DoubleLinkedLists

1. Atribut dari class Node14, yang digunakan untuk menentukan posisi dari node & Konstruktor default untuk menginisialisasi object dari class Node14.

```
public class DoubleLinkedLists {
    Node14 head, tail;

    public DoubleLinkedLists() {
        head = tail = null;
    }
```

2. Method isEmpty(), cek apakah double linked list kosong.

```
public boolean isEmpty() {
    return head == null;
}
```

3. addFirst(), membawa object dari class mahasiswa untuk ditambahkan ke dalam node yang akan dibuat. Node yang dibuat akan dialokasikan ke urutan paling depan dan menjadikannya sebagai head dalam linked list.

```
public void addFirst(Mahasiswa14 data) {
    Node14 newNode = new Node14(data);
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        newNode.next = head;
        head.prev = newNode;
        head = newNode;
    }
}
```

4. addLast(), mengalokasikan node terbaru ke urutan paling belakang dan menjadikannya sebagai tail dalam linked list.

```
public void addLast(Mahasiswa14 data) {
    Node14 newNode = new Node14(data);
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail.next = newNode;
        newNode.prev = tail;
        tail = newNode;
    }
}
```

5. insertAfter(), menambahkan data ke node dengan urutan setelah node yang sesuai dengan kata kunci tertentu, dalam kasus ini kata kunci merujuk pada atribut nama dari class Mahasiswa.

```
public void insertAfter(String keyNim, Mahasiswa14 data) {
    Node14 current = head;

    while (current != null && !current.data.nim.equals(keyNim)) {
        current = current.next;
    }

    if (current == null) {
        System.out.println("Node dengan NIM " + keyNim + " tidak ditemukan.");
        return;
    }

    Node14 newNode = new Node14(data);

    if (current == tail) {
        current.next = newNode;
        newNode.prev = current;
        tail = newNode;
    } else {
        newNode.next = current.next;
        newNode.prev = current;
        current.next.prev = newNode;
        current.next = newNode;
    }
    System.out.println("Node berhasil disisipkan setelah NIM " + keyNim);
}
```

6. Method print(), menampilkan seluruh data yang ada dengan menggunakan object current dengan traverse ke satu node ke node yang lainnya.

```
public void print() {
    Node14 current = head;
    while (current != null) {
        current.data.tampil();
        current = current.next;
    }
}
```

7. Method search(), mencari node yang sesuai menggunakan kata kunci nim.

```
public Node14 search(String nim){
    Node14 current = head;
    while (current != null) {
        if (current.data.nim.equalsIgnoreCase(nim)) {
            return current;
        }
        current = current.next;
    }
    return null;
}
```

D. Class DLLMain

1. Class ini digunakan untuk melakukan tes terhadap method di class DoubleLinkedLists dengan menggunakan simulasi menu.

```
public static void main(String[] args) {
    DoubleLinkedLists list = new DoubleLinkedLists();
    Scanner scan = new Scanner (System.in);
    int pilihan;

    do {
        System.out.println(x:"\nMenu Double Linked List Mahasiswa");
        System.out.println(x:"1. Tambah di awal");
        System.out.println(x:"2. Tambah di akhir");
        System.out.println(x:"3. Hapus diawal");
        System.out.println(x:"4. Hapus di akhir");
        System.out.println(x:"5. Tampilkan data");
        System.out.println(x:"6. Cari Mahasiswa berdasarkan NIM");
        System.out.println(x:"0. Keluar");
        System.out.print(s:"Pilih Menu: "); pilihan = scan.nextInt(); scan.nextLine();

        switch (pilihan) {
         case 1 -> {
            Mahasiswa14 mhs = inputMahasiswa(scan);
            list.addFirst(mhs);
         }
         case 2 -> {
            Mahasiswa14 mhs = inputMahasiswa(scan);
            list.addLast(mhs);
         }
         case 3 -> list.removeFirst();
         case 4 -> list.removeLast();
         case 5 -> list.print();
```

Verifikasi Percobaan

✔️ Hasil sesuai dengan yang ada di jobsheet

```
Menu Double Linked List Mahasiswa
1. Tambah di awal
2. Tambah di akhir
3. Hapus diawal
4. Hapus di akhir
5. Tampilkan data
6. Cari Mahasiswa berdasarkan NIM
0. Keluar
Pilih Menu: 1
Masukkan NIM: 244107020190
Masukkan Nama: Jiro
Masukkan Kelas: 1E
Masukkan IPK: 3,4

Menu Double Linked List Mahasiswa
1. Tambah di awal
2. Tambah di akhir
3. Hapus diawal
4. Hapus di akhir
5. Tampilkan data
6. Cari Mahasiswa berdasarkan NIM
0. Keluar
Pilih Menu: 5
NIM: 244107020190, Nama: Jiro, Kelas: 1E, IPK: 3.4
```

|| Pertanyaan Percobaan 1

1. Jelaskan perbedaan antara single linked list dengan double linked lists!
2. Perhatikan class Node01, di dalamnya terdapat atribut next dan prev. Untuk apakah atribut tersebut?
3. Perhatikan konstruktor pada class DoubleLinkedLists. Apa kegunaan dari konstruktor tersebut?

```
public DoubleLinkedList01() {
    head = null;
    tail = null;
}
```

4. Pada method **addFirst()**, apa maksud dari kode berikut?

```
if (isEmpty()) {
    head = tail = newNode;
```

5. Perhatikan pada method **addFirst().** Apakah arti statement head.prev = newNode ?
6. Modifikasi code pada fungsi **print()** agar dapat menampilkan warning/ pesan bahwa linked lists masih dalam kondisi.

7. Pada insertAfter(), apa maksud dari kode berikut ?

```
current.next.prev = newNode;
```

8. Modifikasi menu pilihan dan switch-case agar fungsi **insertAfter()** masuk ke dalam menu pilihan dan dapat berjalan dengan baik.

Jawaban:

1. Perbedaan utama pada keduanya terlihat pada pointer yang dimiliki. SLL hanya memiliki satu pointer yakni next. Sedangkan DLL memiliki satu pointer tambahan yaitu prev. Dalam proses traverse, SLL hanya akan berjalan terus menuju arah ke depan dari node head hingga menuju node tail, sehingga tak memungkinkan apabila ingin mengubah arahnya selain ke depan. DLL memiliki 2 arah, yakni ke belakang dan ke depan. Memungkinkan untuk melakukan yang tak bisa SLL lakukan. Itu bisa dilakukan karena adanya pointer prev & next pada DLL yang memudahkan proses traverse.

2. Kedua pointer tersebut untuk memudahkan proses traverse diantara node-node yang ada dalam linked list. Traverse bisa dilakukan ke arah belakang (tail menuju head) atau ke depan (head menuju tail).

3. Untuk menginisialisasi nilai dari head & tail menjadi null yang berarti linked list dalam keadaan kosong tak memiliki node.

4. Menginisialisasi nilai head & tail pada node yang ingin dimasukkan saat linked list masih dalam keadaan kosong.

5. Menginisialisasi bahwa node baru akan ditempatkan pada posisi paling depan sehingga node head saat ini perlu menyambung dengan node baru menggunakan head.prev.

6. Modifikasi:

```
public void print() {
    if (isEmpty()) {
        System.out.println(x:"Linked List masih dalam keadaan kosong");
        return;
    } else {
        Node14 current = head;
        while (current != null) {
            current.data.tampil();
            current = current.next;
        }
    }

}
```

7. Kode tersebut bermaksud bahwa pada node A setelah node current akan diinisialisasi pointer prev miliknya untuk menyambung dengan node baru yang akan disimpan. Node A ialah node yang telah ditemukan melalui pencarian dari method terkait.

8. Simulasi

   Sebelum disisipkan:

```
Pilih Menu: 5
NIM: 123, Nama: SQ, Kelas: UA, IPK: 3.9
NIM: RU, Nama: MI, Kelas: NA, IPK: 3.2
```

Memanggil menu:

```
7. Masukkan data setelah urutan node tertentu
0. Keluar
Pilih Menu: 7
-- Mengisi Data --
Masukkan NIM: SI
Masukkan Nama: TY
Masukkan Kelas: XK
Masukkan IPK: 4,3
Masukkan NIM sebagai kata kunci: 123
Node berhasil disisipkan setelah NIM 123
```

Setelah disisipkan:

```
Pilih Menu: 5
NIM: 123, Nama: SQ, Kelas: UA, IPK: 3.9
NIM: SI, Nama: TY, Kelas: XK, IPK: 4.3
NIM: RU, Nama: MI, Kelas: NA, IPK: 3.2
```

2. Percobaan 2

1. Method removeFirst() untuk menghapus node paling depan yakni head.
   Method removeLast() untuk menghapus node paling belakang yakni tail .

```
public void removeFirst(){
    if (isEmpty()) {
        System.out.println(x:"List kosong, tidak bisa dihapus.");
        return;
    }
    if (head == tail) {
        head = tail = null;
    }else{
        head = head.next;
        head.prev = null;
    }
}

public void removeLast(){
    if (isEmpty()) {
        System.out.println(x:"List kosong, tidak bisa dihapus.");
        return;
    }
    if (head == tail) {
        head = tail = null;
    }else {
        tail = tail.prev;
        tail.next = null;
    }
}
```

Verifikasi Percobaan

✔️ Hasil sesuai dengan target method

```
Pilih Menu: 5
NIM: 125, Nama: 125, Kelas: 125, IPK: 125.0
NIM: 124, Nama: 124, Kelas: 124, IPK: 124.0
NIM: 123, Nama: 123, Kelas: 123, IPK: 123.0

Menu Double Linked List Mahasiswa
1. Tambah di awal
2. Tambah di akhir
3. Hapus di awal
4. Hapus di akhir
5. Tampilkan data
6. Cari Mahasiswa berdasarkan NIM
7. Masukkan data setelah urutan node tertentu
0. Keluar
Pilih Menu: 3
```

```
Pilih Menu: 5
NIM: 124, Nama: 124, Kelas: 124, IPK: 124.0
NIM: 123, Nama: 123, Kelas: 123, IPK: 123.0
```

|| Pertanyaan Percobaan 2

1. Apakah maksud statement berikut pada method **removeFirst()**?
   ```
   head = head.next;
   head.prev = null;
   ```
2. Modifikasi kode program untuk menampilkan pesan "Data sudah berhasil dihapus. Data yang terhapus adalah ... "

Jawaban:

1. Kode tersebut ialah bagian untuk menginisialisasi node setelah head untuk dijadikan sebagai head yang baru setelah method dijalankan. Dan node sebelum head dipastikan null karena kini node tersebut berada pada posisi paling depan.

2. Modifikasi:
   Method removeFirst

```
System.out.print(s:"Data ini telah dihapus: "); head.data.tampil();
```

Method removeLast():

```
System.out.print(s:"Data ini telah dihapus: "); tail.data.tampil();
```

3. Latihan Praktikum

12.5 Tugas Praktikum

1. Tambahkan fungsi add() pada kelas DoubleLinkedList untuk menambahkan node pada indeks tertentu
2. Tambahkan removeAfter() pada kelas DoubleLinkedList untuk menghapus node setelah data *key*.
3. Tambahkan fungsi remove() pada kelas DoubleLinkedList untuk menghapus node pada indeks tertentu.
4. Tambahkan fungsi getFirst(), getLast() dan getIndex() untuk menampilkan data pada node head, node tail dan node pada indeks tertentu.
5. tambahkan kode program dan fungsi agar dapat membaca size/ jumlah data pada Double Linked List

--- *** ---

1. addByIndex()

```
public void addByIndex(int cari, Mahasiswa14 data) {
    if (isEmpty()) {
        System.out.println(x:"Tidak ada node yang tercatat.");
        return;
    } else {
        Node14 temp = head;
        int index = 0;
        while (index != cari) {
            temp = temp.next;
            index++;
        }

        if (temp == head) {
            addFirst(data);
            return;
        }
        if (temp == tail) {
            addLast(data);
            return;
        } else {
            Node14 newNode = new Node14(data);
            temp.prev.next = newNode;
            newNode.next = temp;
            temp.prev = newNode;
            size++;
            System.out.println(x:"Data telah masuk !");
        }
    }
}
```

2. removeAfter()

```java
public void removeAfter(String keyNim) {
    Node14 current = head;

    while (current != null && !current.data.nim.equals(keyNim)) {
        current = current.next;
    }

    if (current == null || current.next == null) {
        System.out.println("Node dengan NIM " + keyNim + " tidak ditemukan.");
        return;
    }
    if (current.next == tail) {
        removeLast();
    } else {
        current.next = current.next.next;
        current.next.prev = current;
    }
    size--;
    System.out.printf(format:"Node setelah NIM %s telah dihapus", keyNim);
}
```

3. removeByIndex()

```java
public void removeByIndex(int cari) {
    if (isEmpty()) {
        System.out.println(x:"Tidak ada node yang tercatat.");
        return;
    } else {
        Node14 temp = head;
        int index = 0;
        while (index != cari) {
            temp = temp.next;
            index++;
        }
        if (temp == tail) {
            removeLast();
            return;
        } else {
            temp.prev.next = temp.next;
            temp.next.prev = temp.prev;
            size--;
            System.out.println(x:"Data telah dihapus !");
        }
    }
}
```

4. getFirst(), getLast(), getIndex()

```
public void getFirst() {
    head.data.tampil();
}

public void getLast() {
    tail.data.tampil();
}

public void getIndex(int cari) {
    int index = 0;
    Node14 sweep = head;
    while (index != cari) {
        sweep = sweep.next;
        index++;
    } sweep.data.tampil();
}
```

5. getSize()

```
public void getSize() {
    System.out.printf(format:"Jumlah data/size saat ini: %d%n", size);
}
```