Laporan Jobsheet 10

Algoritma dan Struktur Data

Disusun Oleh :

DHANISA PUTRI MASHILFA

NIM. 2341720212

TI-1E/07

D-IV TEKNIK INFORMATIKA

JURUSAN TEKNOLOGI INFORMASI

POLITEKNIK NEGERI MALANG

Jl. Soekarno Hatta No. 9 Jatimulyo, Kecamatan Lowokwaru, Jatimulyo, Kec. Lowokwaru, Kota Malang, Jawa Timur 6514

10.2.1 Percobaan Praktikum 1

→ Node.java

```
public class Node {

    int data;
    Node prev;
    Node next;

    Node (Node prev, int data, Node next) {

        this.prev = prev;
        this.data = data;
        this.next = next;
    }
}
```

→ DoubleLinkedList.java

```
public class DoubleLinkedList {

    Node head;
    int size;

    public DoubleLinkedList() {
        head = null;
        size = 0;
    }

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

    public void addFirst(int item) {
        if (isEmpty()) {
            head = new Node (null, item, null);
        } else {
            Node newNode = new Node (null, item, head);
            head.prev = newNode;
            head = newNode;
        }
        size++;
    }

    public void addLast(int item) {
        if (isEmpty()) {
            addFirst(item);
        } else {
            Node current = head;
            while (current.next != null) {
                current = current.next;
            }
            Node newNode = new Node(current, item, null);
            current.next = newNode;
            size++;
```

```
        }
    }

    public void add(int item, int index) throws Exception {
        if (isEmpty()) {
            addFirst (item);
        } else if (index < 0 || index > size) {
            throw new Exception("Nilai indeks di luar batas");
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.prev == null) {
                Node newNode = new Node (null, item, current);
                current.prev = newNode;
                head= newNode;
            } else {
                Node newNode = new Node (current.prev, item, current);
                newNode.prev = current.prev;
                newNode.next = current;
                current.prev.next = newNode;
                current.prev = newNode;
            }
        }
        size++;
    }

    public int size() {
        return size;
    }

    public void clear() {
        head = null;
        size = 0;
    }

    public void print() {
        if (!isEmpty()) {
            Node tmp = head;
            while (tmp != null) {
                System.out.print(tmp.data + "\t");
                tmp = tmp.next;
            }
            System.out.println("\nBerhasil diisi");
        } else {
             System.out.println();
            System.out.println("Linked List Kosong");
        }
    }
}
```

→ **DoubleLinkedListMain.java**

```java
public class DoubleLinkedListMain {

    public static void main(String[] args) {

        DoubleLinkedList dll = new DoubleLinkedList();
        dll.print();

        System.out.println("Size : " + dll.size);
        System.out.println("=====================================");
        dll.addFirst(3);
        dll.addLast(4);
        dll.addFirst(7);
        dll.print();

        System.out.println("Size : " + dll.size());
        System.out.println("=====================================");
        try {
            dll.add(40, 1);
        } catch (Exception e) {
            System.out.println(e);
        }
        dll.print();

        System.out.println("Size : " + dll.size);
        System.out.println("=====================================");
        dll.clear();
        dll.print();

        System.out.println("Size : " + dll.size);
    }
}
```

10.2.2 Verifikasi hasil Percobaan 1

```
--- exec-maven-plugin:1.5.0:exec
Linked Lists Kosong
Size: 0
=======================
7       3       4
berhasil diisi
Size: 3
=======================
7       40      3       4
berhasil diisi
Size: 4
=======================
Linked Lists Kosong
Size: 0
=======================
------------------------------------
BUILD SUCCESS
------------------------------------
```

```
Linked List Kosong
Size : 0
=====================================
7       3       4
Berhasil diisi
Size : 3
=====================================
7       40      3       4
Berhasil diisi
Size : 4
=====================================

Linked List Kosong
Size : 0
PS C:\Users\asus\Documents\Semester2\A
```

10.2.3 Pertannyaan Percobaan 1

1. Jelaskan perbedaan antara single linked list dengan double linked lists!
   → Perbedaan antara single linked list dengan double linked lists salah satu diantaranya terdapat dalam struktur **Node.**

   Pada Linked List Setiap node dalam single linked list hanya memiliki satu pointer yang mengarah ke node berikutnya. Sedangkan, Setiap node dalam double linked list memiliki dua pointer: satu mengarah ke node berikutnya dan satu lagi mengarah ke node sebelumnya.

2. Perhatikan class Node, di dalamnya terdapat atribut next dan prev. Untuk apakah atribut tersebut?
   → Atribut **prev** digunakan untuk menyimpan alamat dari node sebelumnya dalam linked list, serta dapat digunakan untuk traversal ke arah belakang dalam linked list.

3. Perhatikan konstruktor pada class DoubleLinkedLists. Apa kegunaan inisialisasi atribut head dan size seperti pada gambar berikut ini?

```
public DoubleLinkedLists() {
    head = null;
    size = 0;
}
```

   → menginisialisasikan **head** ke **null** untuk menunjukan jika listnya kosong di awal dan menginisialisasikan **size** ke **0** untuk menunjukan bahwa tidak ada elemen dalam list.

4. Pada method **addFirst()**, kenapa dalam pembuatan object dari konstruktor class Node prev dianggap sama dengan null? Node newNode = new Node(null, item, head);
   → Di dalam list node baru akan menjadi node pertama. Sehingga, **prev** diatur ke **null** agar tidak ada node sebelum itu.

5. Perhatikan pada method **addFirst().** Apakah arti statement head.prev = newNode ?
   → Menghubungkan node baru dengan node lama, **prev** diperbarui dari node yang saat ini ke **head** untuk menunjuk ke node baru.

6. Perhatikan isi method **addLast()**, apa arti dari pembuatan object Node dengan mengisikan parameter prev dengan current, dan next dengan null?
   Node newNode = new Node**(current, item, null);**
   a. **current** : Untuk mengatur node saat ini sebagai **prev** karena node baru akan mengikuti node terakhir.
   b. **item** : Data yang disimpan dalam node baru.
   c. **null** : untuk **next** karena tidak ada node berikutnya karena node baru akan menjadi node terakhir.

7. Pada method **add()**, terdapat potongan kode program sebagai berikut:

```
while (i < index) {
    current = current.next;
    i++;
}
if (current.prev == null) {
    Node newNode = new Node(null, item, current);
    current.prev = newNode;
    head = newNode;
} else {
    Node newNode = new Node(current.prev, item, current);
    newNode.prev = current.prev;
    newNode.next = current;
    current.prev.next = newNode;
    current.prev = newNode;
}
```

   jelaskan maksud dari bagian yang ditandai dengan kotak kuning.
   → digunakan untuk menangani situasi tertentu di mana perlu menambahkan node baru di depan node, atau node pertama dari linked list. Ini memastikan bahwa pointer prev dan next diatur dengan benar, menjaga integritas struktur double linked list dan memperbarui kepala untuk menunjuk ke node baru.

10.3.1 Percobaan Praktikum 2

→ DoubleLinkedList.java

```
public class DoubleLinkedList {

    Node head;
    int size;

    public void removeFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked List masih kosong, tidak dapat
dihapus!");
        } else if (size == 1) {
            removeFirst();
        } else {
            head = head.next;
            head.prev = null;
            size--;
        }
    }

    public void removeLast() throws Exception {
        if (isEmpty()) {
            throw new Exception ("Linked List masih kosong, tidak dapat
dihapus");
        } else if(head.next == null) {
            head = null;
            size--;
            return;
        }
        Node current = head;
        while (current.next.next != null) {
            current = current.next;
        }
        current.next = null;
        size--;
    }

    public void remove(int index) throws Exception {
        if (isEmpty()) {
            throw new Exception ("Nilai indeks di luar batas");
        } else if (index == 0) {
            removeFirst();
        } else {
            Node current = head;
            int i = 0;
            while (i < index) {
                current = current.next;
                i++;
            }
            if (current.next == null) {
                current.prev.next = null;
            } else if (current.prev == null) {
                current = current.next;
```

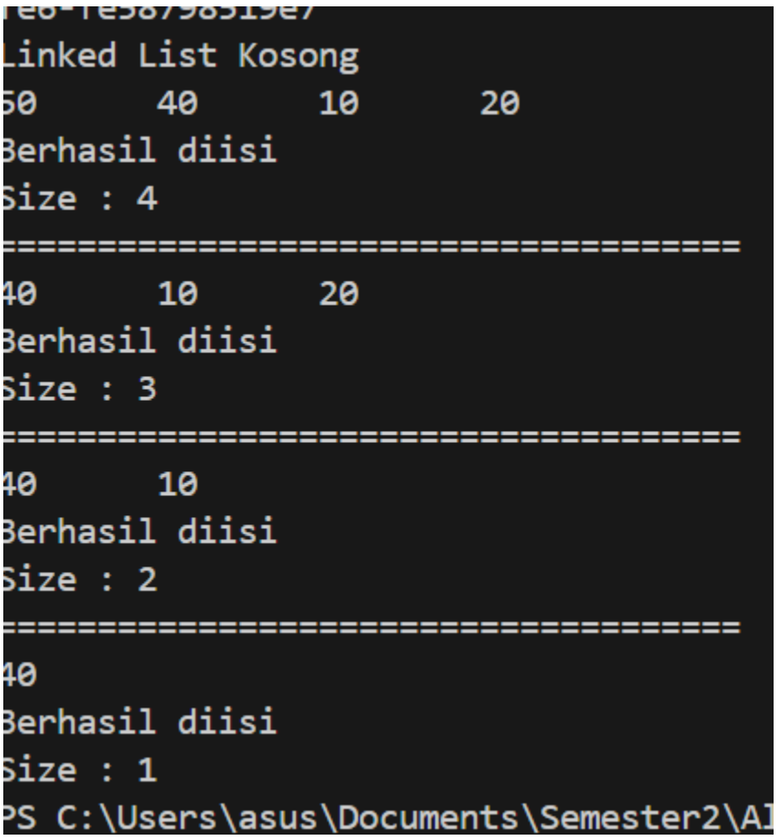
```
                current.prev = null;
                head = current;
            } else {
                current.prev.next = current.next;
                current.next.prev = current.prev;
            }
            size--;
        }
    }
}
```

→ DoubleLinkedListMain.java

```
public class DoubleLinkedListMain {

    public static void main(String[] args) throws Exception {

        DoubleLinkedList dll = new DoubleLinkedList();
        dll.print();

        dll.addLast(50);
        dll.addLast(40);
        dll.addLast(10);
        dll.addLast(20);
        dll.print();
        System.out.println("Size : " + dll.size);
        System.out.println("=====================================");

        dll.removeFirst();
        dll.print();
        System.out.println("Size : " + dll.size);
        System.out.println("=====================================");

        dll.removeLast();
        dll.print();
        System.out.println("Size : " + dll.size);
        System.out.println("=====================================");
        dll.remove(1);
        dll.print();

        System.out.println("Size : " + dll.size);
    }
}
```

10.3.2 Verifikasi hasil Percobaan 2

```
--- exec-maven-plugin:1.5.0:exec
50      40      10      20
berhasil diisi
Size: 4
=======================
40      10      20
berhasil diisi
Size: 3
=======================
40      10
berhasil diisi
Size: 2
=======================
40
berhasil diisi
Size: 1
------------------------------
BUILD SUCCESS
------------------------------
```



10.3.3 Pertanyaan Percobaan 2

1. Apakah maksud statement berikut pada method **removeFirst()?**
   head = head.next;
   → Menghapus node pertama dari linked list secara efektif, **head** digunakan untuk menunjuk ke node berikutnya.
   head.prev = null;
   → Mengatur **prev** dari node **head** baru ke **null** untuk menghindari referensi kembali ke node yang dihapus.

2. Bagaimana cara mendeteksi posisi data ada pada bagian akhir pada method **removeLast()?**
   → Mencari data hingga node terakhir dan menghapusnya dengan mengatur next dari node sebelumnya menjadi null.

3. Jelaskan alasan potongan kode program di bawah ini tidak cocok untuk perintah **remove!**

```
Node tmp = head.next;

head.next=tmp.next;
tmp.next.prev=head;
```

   → Potongan kode tersebut tidak cocok karena teknik tersebut digunakan untuk linkedlist, apabila dalam double linkedlist menggunakan kode tersebut akan terjadi beberapa masalah. Salah satu masalahnya adalah kesalahan dalam menghapus node pertama.

4. Jelaskan fungsi kode program berikut ini pada fungsi **remove!**

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

   → Untuk menghapus node dari linked list, metode **remove()** digunakan. Untuk melakukan ini, pointer **next** dari node sebelumnya dan pointer **prev** dari node setelahnya diubah, ini menghentikan koneksi langsung antara node yang akan dihapus dan node tetangganya, yang berarti node tersebut dihapus dari daftar terhubung. Ini adalah langkah penting untuk memastikan struktur linked list konsisten dan konsisten setelah penghapusan.

10.4.1 Percobaan Praktikum 3

→ DoubleLinkedList.java

```
public class DoubleLinkedList {

    Node head;
    int size;

    public int getFirst() throws Exception {
        if (isEmpty()) {
            throw new Exception ("Linked List kosong");
        }
        return head.data;
    }

    public int getLast() throws Exception {
        if (isEmpty()) {
            throw new Exception("Linked List kosng");
        }
        Node tmp = head;
        while (tmp.next != null) {
            tmp = tmp.next;
        }
        return tmp.data;
    }

    public int get(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Nilai indeks di luar batas.");
        }
        Node tmp = head;
        for (int i = 0; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }
}
```

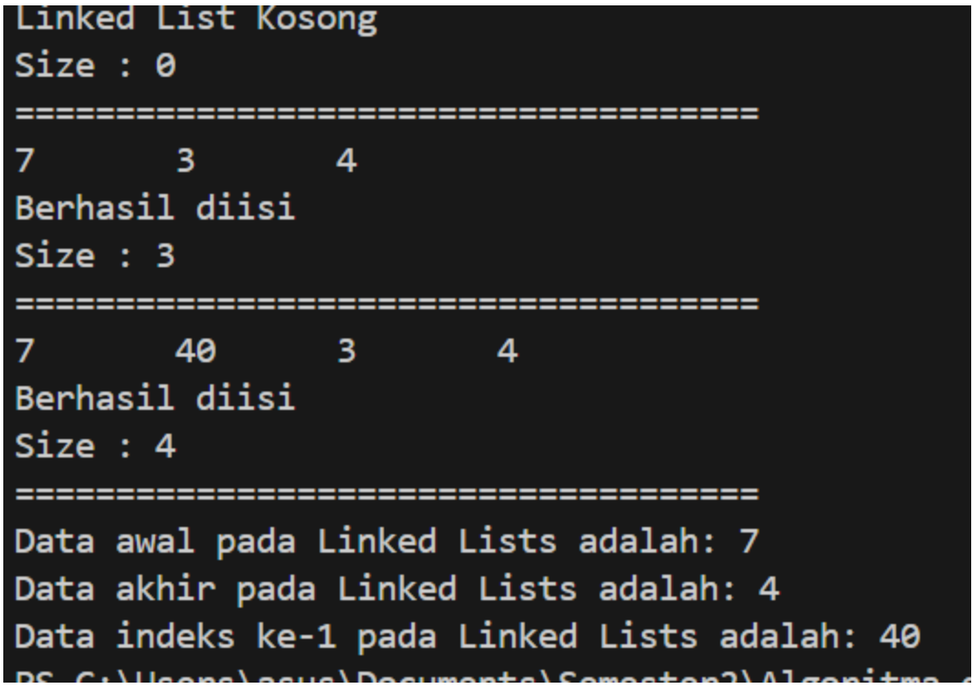
→ DoubleLinkedListMain.java

```
public class DoubleLinkedListMain {

    /**
     * @param args
     * @throws Exception
     */
    public static void main(String[] args) throws Exception {

        DoubleLinkedList dll = new DoubleLinkedList();
        dll.print();

        System.out.println("Size : " + dll.size);
        System.out.println("====================================");
        dll.addFirst(3);
```

```
        dll.addLast(4);
        dll.addFirst(7);
        dll.print();

        System.out.println("Size : " + dll.size);
        System.out.println("=======================================");
        dll.add(40, 1);
        dll.print();

        System.out.println("Size : " + dll.size);
        System.out.println("=======================================");
        System.out.println("Data awal pada Linked Lists adalah: " +
dll.getFirst());
        System.out.println("Data akhir pada Linked Lists adalah: "+
dll.getLast());
        System.out.println("Data indeks ke-1 pada Linked Lists adalah: " +
dll.get(1));
    }
}
```

10.4.2 Verifikasi hasil Percobaan 3

```
--- exec-maven-plugin:1.5.0:exec (default-cli)
Linked Lists Kosong
Size: 0
=======================
7       3       4
berhasil diisi
Size: 3
=======================
7       40      3       4
berhasil diisi
Size: 4
=======================
Data awal pada Linked Lists adalah: 7
Data akhir pada Linked Lists adalah: 4
Data indeks ke-1 pada Linked Lists adalah: 40
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```



10.4.3 Pertanyaan Percobaan 3

1. Jelaskan method size() pada class DoubleLinkedLists!
   → metode sederhana untuk mengembalikan jumlah elemen dalam linked list saat itu. Ini adalah fitur penting dalam pengelolaan linked list karena memungkinkan pengguna untuk mengetahui ukuran atau panjang linked list, yang penting untuk berbagai operasi dan kebutuhan pemrograman.

2. Jelaskan cara mengatur indeks pada double linked lists supaya dapat dimulai dari indeks ke 1!
   → Pada method get() kode programnya di modifikasi

```
public int get(int index) throws Exception {
        if (isEmpty() || index >= size) {
            throw new Exception("Nilai indeks di luar batas.");
        }
        Node tmp = head;
        for (int i = 1; i < index; i++) {
            tmp = tmp.next;
        }
        return tmp.data;
    }
```

```
Size : 0
==================================
7       3       4
Berhasil diisi
Size : 3
==================================
7       40      3       4
Berhasil diisi
Size : 4
==================================
Data awal pada Linked Lists adalah: 7
Data akhir pada Linked Lists adalah: 4
Data indeks ke-1 pada Linked Lists adalah: 7
```

3. Jelaskan perbedaan karakteristik fungsi Add pada Double Linked Lists dan Single Linked Lists!
   → Perbedaan utama antara karakteristik fungsi add pada double linked list dan single linked list terletak pada efisiensi operasi penambahan di akhir list dan di tengah list. Double linked list lebih efisien dalam hal ini karena struktur data yang lebih kompleks, yang mengakibatkan operasi penambahan memiliki kompleksitas waktu yang lebih rendah, terutama untuk penambahan di akhir list.

4. Jelaskan perbedaan logika dari kedua kode program di bawah ini!

```java
public boolean isEmpty(){
    if(size ==0){
        return true;
    } else{
        return false;
    }
}
```

(a)

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

(b)

   → Tujuan kedua metode sama, tetapi logikanya sedikit berbeda. Metode pertama memeriksa keberadaan pointer head dan lebih sederhana dan efektif, tetapi metode kedua menggunakan variabel ukuran untuk memeriksa ukuran daftar, yang membutuhkan alokasi memori tambahan dan penyesuaian setiap kali terjadi perubahan pada daftar yang terhubung.