JOBSHEET IX

LINKED LIST

1. Tujuan Praktikum

Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Membuat struktur data linked list
2. Membuat linked list pada program
3. Membedakan permasalahan apa yang dapat diselesaikan menggunakan linked list

2. Praktikum

2.1 Pembuatan Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, akan dilakukan implementasi pembuatan linked list menggunakan array dan penambahan node ke dalam linked list

1. Buat folder baru Praktikum09
2. Tambahkan class-class berikut:
    a. Node.java
    b. LinkedList.java
    c. SLLMain.java
3. Deklarasikan class Node yang memiliki atribut data untuk menyimpan elemen dan atribut next bertipe Node untuk menyimpan node berikutnya. Tambahkan constructor berparameter untuk mempermudah inisialisasi

```
public class Node {
    int data;
    Node next;

    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
}
```

4. Deklarasikan class LinkedList yang memiliki atribut `head`. Atribut head menyimpan node pertama pada linked list

```
public class LinkedList {
    Node head;
}
```

5. Sebagai langkah berikutnya, akan diimplementasikan method-method yang terdapat pada class LinkedList.
6. Tambahkan method **isEmpty()**

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

7. Implementasi method print() untuk mencetak dengan menggunakan proses traverse.

```
public void print() {
    if (!isEmpty()) {
        System.out.print("Isi linked list: ");
        Node currentNode = head;

        while (currentNode != null) {
            System.out.print(currentNode.data + "\t");
            currentNode = currentNode.next;
        }

        System.out.println("");
    } else {
        System.out.println("Linked list kosong");
    }
}
```

8. Implementasikan method **addFirst()** untuk menambahkan node baru di awal linked list

```
public void addFirst(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
}
```

9. Implementasikan method **addLast()** untuk menambahkan node baru di akhir linked list

```
public void addLast(int input) {
    Node newNode = new Node(input, null);

    if (isEmpty()) {
        head = newNode;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            currentNode = currentNode.next;
        }

        currentNode.next = newNode;
    }
}
```

10. Implementasikan method **insertAfter()** menambahkan node baru pada posisi setelah node yang berisi data tertentu (key)

```
public void insertAfter(int key, int input) {
    Node newNode = new Node(input, null);

    if (!isEmpty()) {
        Node currentNode = head;

        do {
            if (currentNode.data == key) {
                newNode.next = currentNode.next;
                currentNode.next = newNode;
                break;
            }

            currentNode = currentNode.next;
        } while (currentNode != null);
    } else {
        System.out.print("Linked list kosong");
    }
}
```

11. Pada class SLLMain, buatlah fungsi **main**, kemudian buat object myLinkedList bertipe LinkedList. Lakukan penambahan beberapa data. Untuk melihat efeknya terhadap object myLinkedList, panggil method print()

```
    public static void main(String[] args) {
        LinkedList myLinkedList = new LinkedList();
        myLinkedList.print();
        myLinkedList.addFirst(800);
        myLinkedList.print();
        myLinkedList.addFirst(700);
        myLinkedList.print();
        myLinkedList.addLast(500);
        myLinkedList.print();
        myLinkedList.insertAfter(700, 300);
        myLinkedList.print();
    }
```

2.1.1 Verifikasi Hasil Percobaan

Cocokkan hasil run program Anda dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
```

```
Praktikum09 > J LinkedList.java > LinkedList > addLast(int)
1   public class LinkedList {
2       Node head;
3       public boolean isEmpty() {
4           return (head == null);
5       }
6       public void print(){
7           if (!isEmpty()) {
8               System.out.print(s:"Isi linked list: ");
9               Node currentNode = head;
10
11              while (currentNode != null ) {
12                  System.out.print(currentNode.data + "\t");
13                  currentNode = currentNode.next;
14              }
15              System.out.println(x:"");
16          }else {
17              System.out.println(x:"Linked list kosong");
18          }
19      }
20          public void addFirst (int input) {
21              Node newNode = new Node (input, next:null);
22
23              if (isEmpty()) {
24                  head = newNode;
25              }else {
26                  newNode.next = head;
27                  head = newNode;
28              }
29          }
30          public void addLast (int input) {
31              Node newNode = new Node(input, next:null);
32
33              if (isEmpty()) {
34                  head = newNode;
35              }else {
36                  Node currentNode = head;
37
38                  while (currentNode.next != null){
39                      currentNode = currentNode.next;
40                  }
41                  currentNode.next = newNode;
42              }
43          }
44          public  void insertAfter (int key, int input){
45              Node newNode = new Node(input, next:null);
```

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
```

2.1.2 Pertanyaan

1. Mengapa class LinkedList tidak memerlukan method isFull() seperti halnya Stack dan Queue?

    - Class Linkedlist tidak memerlukan method isFull( ) seperti Stack dan Queue karena sifat dasar dari struktur data LinkedList.

2. Mengapa class LinkedList hanya memiliki atribut head yang menyimpan informasi node pertama? Bagaimana informasi node kedua dan lainnya diakses?
    - Class LinkedList hanya memiliki atribut head yang menyimpan informasi node pertama karena sifat dasar dari struktur data LinkedList.
    - Untuk mengakses node kedua dan selanjutnya dalam daftar, kita dapat mengikuti pointer next dari node pertama. Berikut langkah2nya:
        a. Memulai dari head : kita memulai dengan node pertama yang diwakili oleh atribut head
        b. Mengikuti pointer next : Kita mengikuti pointer next dari node pertama untuk mencapai node kedua
        c. Iterasi berulang : kita dapat terus mengikuti pointer next dari node saat ini untuk mencapai node berikutnya dan seterusnya.
3. Pada langkah, jelaskan kegunaan kode berikut

```
if (currentNode.data == key) {
    newNode.next = currentNode.next;
    currentNode.next = newNode;
    break;
}
```

    - Kode diatas merupakan bagian dari implementasi operasi pencarian dalam struktur data LinkedList. Kode ini mengecek apakah elemen dengan nilai tertentu (key) terdapat dalam daftar LinkedList.

4. Implementasikan method insertAt(int index, int key) dari tugas mata kuliah ASD (Teori)

```
public void insertArt (int index, int key){
    Node10 newNode = new Node10(key, next:null);
    if (index < 0){
        System.out.println(x:"Index tidak valid");
    }else if (index == 0) {
        newNode.next = head;
        head = newNode;
    }else{
        Node10 current = head;
        int currentIndex = 0;
        while (currentIndex < index -1){
            current = current.next;
            currentIndex++;
        }
        if (current == null){
            System.out.println(x:"Index melebihi batas");
        } else {
            newNode.next = current.next;
            current.next = newNode;
        }
    }
}
```

2.2 Mengakses dan menghapus node pada Linked List

Waktu percobaan: 50 menit

Didalam praktikum ini, kita akan mengimplementasikan method untuk melakukan pengaksesan dan penghapusan data pada linked list

2.2.1 Langkah-langkah Percobaan

1. Tambahkan method getData() untuk mengembalikan nilai elemen di dalam node pada index tertentu

```
public int getData(int index) {
    Node currentNode = head;

    for (int i = 0; i < index; i++) {
        currentNode = currentNode.next;
    }

    return currentNode.data;
}
```

2. Tambahkan method indexOf() untuk mengetahu index dari node dengan elemen tertentu

```
public int indexOf(int key) {
    Node currentNode = head;
    int index = 0;

    while (currentNode != null && currentNode.data != key) {
        currentNode = currentNode.next;
        index++;
    }

    if (currentNode == null) {
        return -1;
    } else {
        return index;
    }
}
```

3. Tambahkan method removeFirst() untuk menghapus node pertama pada linked list

```
public void removeFirst() {
    if (!isEmpty()) {
        head = head.next;
    } else {
        System.out.println("Linked list kosong");
    }
}
```

4. Tambahkan method removeLast() untuk menghapus node terakhir pada linked list

```
public void removeLast() {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.next == null) {
        head = null;
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.next == null) {
                currentNode.next = null;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

5. Method remove() digunakan untuk mengapus node yang berisi elemen tertentu

```
public void remove(int key) {
    if (isEmpty()) {
        System.out.println("Linked list kosong");
    } else if (head.data == key) {
        removeFirst();
    } else {
        Node currentNode = head;

        while (currentNode.next != null) {
            if (currentNode.next.data == key) {
                currentNode.next = currentNode.next.next;
                break;
            }

            currentNode = currentNode.next;
        }
    }
}
```

6. Kemudian, coba lakukan pengaksesan dan penghapusan data di method main pada class SLLMain dengan menambahkan kode berikut

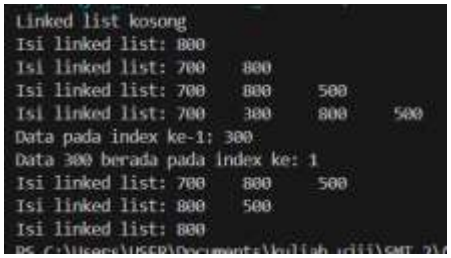
```
System.out.println("Data pada index ke-1: " + myLinkedList.getData(1));
System.out.println("Data 300 berada pada index ke: " + myLinkedList.indexOf(300));

myLinkedList.remove(300);
myLinkedList.print();
myLinkedList.removeFirst();
myLinkedList.print();
myLinkedList.removeLast();
myLinkedList.print();
```

7. Compile dan run program kemudian amati hasilnya

2.2.2 Verifikasi Hasil Percobaan

Cocokkan hasil run program dengan output berikut ini.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 300
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```



2.2.3 Pertanyaan

1. Jelaskan maksud potongan kode di bawah pada method remove()

```
if (currentNode.next.data == key) {
    currentNode.next = currentNode.next.next;
    break;
}
```

   - Kode diatas merupakan bagian dari implementasi operasi penghapusan elemen dalam struktur data LinkedList. Kode ini menghapus elemen dengan nilai tertentu (key) dari daftar LinkedList.

2. Jelaskan maksud if-else block pada method indexOf() berikut

```
if (currentNode == null) {
    return -1;
} else {
    return index;
}
```

   - Kode diatas merupakan bagian dari implementasi metode indexOf( ) dalam struktur data LinkedList. Metode ini mencari indeks dari elemen dengan nilai tertentu (key) dalam daftar LinkedList.

3. Error apa yang muncul jika argumen method getData() lebih besar dari jumlah node pada linked list? Modifikasi kode program untuk menghandle hal tersebut.

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 700
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```

4. Apa fungsi keyword break pada method remove()? Bagaimana efeknya jika baris tersebut dihapus?

```
Linked list kosong
Isi linked list: 800
Isi linked list: 700    800
Isi linked list: 700    800     500
Isi linked list: 700    300     800     500
Data pada index ke-1: 700
Data 300 berada pada index ke: 1
Isi linked list: 700    800     500
Isi linked list: 800    500
Isi linked list: 800
```

3. Tugas

Waktu pengerjaan: 50 menit

1. Implementasikan method-method berikut pada class LinkedList:
   a. insertBefore() untuk menambahkan node sebelum keyword yang diinginkan

```
public void insertBefore(int key, int input) {
    Node10 newNode = new Node10 (input, next:null);
    if (!isEmpty()) {
        if(head.data == key){
            newNode.next = head;
            head = newNode;
        }else {
            Node10 currentNode = head;
            Node10 prevNode = null;
            do{
                if (currentNode.data == key) {
                    newNode.next = currentNode;
                    prevNode.next = newNode;
                    break;
                }
                prevNode = currentNode;
                currentNode = currentNode.next;
            } while (currentNode != null);
        }

    } else{
        System.out.println(x:"Linked List Kosong");
    }
}
```

   b. insertAt(int index, int key) untuk menambahkan node pada index tertentu

```
public void insertArt (int index, int key){
    Node10 newNode = new Node10(key, next:null);
    if (index < 0){
        System.out.println(x:"Index tidak valid");
    }else if (index == 0) {
        newNode.next = head;
        head = newNode;
    }else{
        Node10 current = head;
        int currentIndex = 0;
        while (currentIndex < index -1){
            current = current.next;
            currentIndex++;
        }
        if (current == null){
            System.out.println(x:"Index melebihi batas");
        } else {
            newNode.next = current.next;
            current.next = newNode;
        }
    }
}
```

c. removeAt(int index) untuk menghapus node pada index tertentu

```
public void removeAt (int index) {
    if (isEmpty()){
        System.out.println(x:"Linked List Kosong");
    }else if (index == 0){
        removeFirst();
    } else{
        Node10 currentNode = head;
        int listIndex = 0;
        while(currentNode.next != null && listIndex < index){
            if (listIndex == index -1){
                currentNode.next = currentNode.next.next;
                break;
            }
            currentNode = currentNode.next;
            listIndex++;
        }
    }
}
```

2. Dalam suatu game scavenger hunt, terdapat beberapa point yang harus dilalui peserta untuk menemukan harta karun. Setiap point memiliki soal yang harus dijawab, kunci

ScavengerHunt.java

```
import java.util.Scanner;

public class ScavengerHunt {
    NodeTugas10 head;
    NodeTugas10 tail;

    public ScavengerHunt() {
        this.head = null;
        this.tail = null;
    }
    public void addPoint(String pertanyaan, String jawaban) {
        NodeTugas10 newNode = new NodeTugas10(pertanyaan, jawaban);
        if (head == null) {
            head = newNode;
            tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }
    public void startHunt() {
        Scanner scanner = new Scanner(System.in);
        NodeTugas10 current = head;
        while (current != null) {
            System.out.println(current.pertanyaan);
            String userJawaban = scanner.nextLine();
            if (userJawaban.equalsIgnoreCase(current.jawaban)) {
                System.out.println(x:"Benar! Silahkan lanjut ke pertanyaan selanjutnya.");
                current = current.next;
                if (current == null) {
                    System.out.println(x:"Selamat! Kamu telah menemukan Treasure!");
                }
            } else {
                System.out.println(x:"Jawabanmu salah. Ayo coba lagi.");
            }
        }
        scanner.close();
    }
}
```

Node.java

```java
public class NodeTugas10 {
    String pertanyaan;
    String jawaban;
    NodeTugas10 next;

    NodeTugas10(String pertanyaan, String jawaban) {
        this.pertanyaan = pertanyaan;
        this.jawaban = jawaban;
        this.next = null;
    }
}
```

Main

```java
import java.util.Scanner;
public class ScavengerHuntMain {
    Run | Debug
    public static void main(String[] args) {
        ScavengerHunt game = new ScavengerHunt();
        game.addPoint(pertanyaan:"Dimanakah letak Gunung Semeru?", jawaban:"Lumajang");
        game.addPoint(pertanyaan:"Berpakah hasil 8 x 7", jawaban:"56");
        game.addPoint(pertanyaan:"Apa bahasa inggris buah?", jawaban:"Fruit");


        game.startHunt();
    }
}
```

Output:

```
Dimanakah letak Gunung Semeru?
Lumajang
Benar! Silahkan lanjut ke pertanyaan selanjutnya.
Berpakah hasil 8 x 7
56
Benar! Silahkan lanjut ke pertanyaan selanjutnya.
Apa bahasa inggris buah?
Fruit
Benar! Silahkan lanjut ke pertanyaan selanjutnya.
Selamat! Kamu telah menemukan Treasure!
PS C:\Users\USER\Documents\kuliah ujii\SMT 2\ALGORITMA & STRUKTUR DATA\Praktikum09>
```

jawaban, dan pointer ke point selanjutnya. Buatlah implementasi game tersebut dengan linked list.