JOBSHEET IX
LINKED LIST

Satria Wiguna/Ti 1D/Absen 26

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 Single Linked List

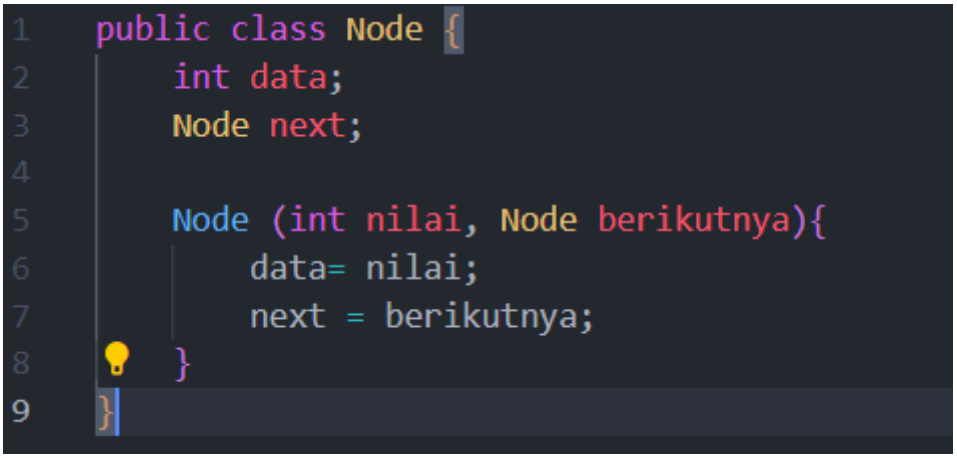
Waktu percobaan : 30 menit

Didalam praktikum ini, kita akan mempraktekkan bagaimana membuat Single Linked List dengan representasi data berupa Node, pengaksesan linked list dan metode penambahan data.

1. Pada Project **StrukturData** yang sudah dibuat pada Minggu sebelumnya, buat package dengan nama **minggu11**
2. Tambahkan class-class berikut:
   a. `Node.java`
   b. `SingleLinkedList.java`
   c. `SLLMain.java`
3. Implementasi class Node

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

    Node(int nilai, Node berikutnya){
        data = nilai;
        next = berikutnya;
    }
}
```



4. Tambahkan atribut pada class SingleLinkedList

```
Node head, tail;
```

```
public class SingleLinkedList {
    Node head,tail;
```

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

```
boolean isEmpty(){ // kondisinya headnya harus berisi null
    return head != null;
}
```

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

7. Implementasi method untuk mencetak dengan menggunakan proses traverse.

```
void print(){ // pencetakan data ini tidak memperbolehkan LL dalam
              // kondisi kosong
    if(isEmpty()){
        Node tmp = head;
        System.out.println("Isi Linked List");
        while(tmp == null){
            System.out.println(tmp.data + "\t");
            tmp = tmp.next;
        }
        System.out.println("");

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

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

8. Implementasikan method **`addFirst()`**.

```
void addFirst(int input){
    // node baru yang ditambahkan berisi data melalui parameter
    // pada method addFirst
    Node ndInput = new Node(input, null);
    if(isEmpty()){ // jika kosong, maka peran head dan tail
                   // harus dimiliki node yang sama
        head = ndInput;
        tail = ndInput;
        ndInput.next = head;
        head = ndInput;
    } else{
        head = ndInput;
        tail = ndInput;
        ndInput.next = head;
        head = ndInput;
    }
}
```

```
void addFirst(int input) {
    Node ndinput = new Node(input, berikutnya:null);
    if (isEmpty()) {
        head = ndinput;
        tail = ndinput;
    } else {
        ndinput.next = head;
        head = ndinput;
    }
}
```

9. Implementasikan method **`addLast()`**.

```
void addLast(int input){
    // node baru yang ditambahkan berisi data melalui parameter
    // pada method addLast
    Node ndInput = new Node();
    if(isEmpty()){ // jika kosong, maka peran head dan tail
                   // harus dimiliki node yang sama
        tail.next = ndInput;
        tail = ndInput;
    } else{
        head = ndInput;
        tail = ndInput;
    }
}
```

```
public void addLast(int input) {
    Node newNode = new Node(input, berikutnya:null);
    if (isEmpty()) {
        head = newNode;
        tail = newNode;
    } else {
        tail.next = newNode;
        tail = newNode;
    }
}
```

10. Implementasikan method **`insertAfter`**, untuk memasukkan node yang memiliki data `input` setelah node yang memiliki data `key`.

```
void insertAfter(int key, int input){
    Node ndInput = new Node();
    Node temp = head;
    do{
        if(temp.data == key){
            ndInput.next= temp.next;
            temp.next = ndInput;
            if(ndInput.next != null){ // jika tidak ada node selanjutnya
                                      // maka jadikan ndInput sebagai tail
                tail=ndInput;
                break; // jangan lupa di rem, jangan gas terus!
            }

        }
        temp = temp.next;
    } while(temp == null); // selama masih ada node, lanjutkan
}
```

```
public void insertAfter(int key, int input) {
    Node newNode = new Node(input, berikutnya:null);
    Node temp = head;
    while (temp != null) {
        if (temp.data == key) {
            newNode.next = temp.next;
            temp.next = newNode;
            if (newNode.next == null) {
                tail = newNode;
            }
            break;
        }
        temp = temp.next;
    } while(temp==null);
}
```

11. Tambahkan method penambahan node pada indeks tertentu.

```
void insertAt(int index, int input){
    // pastikan operasi dari method ini adalah menggeser posisi
    // node yang terletak di indeks dan node tersebut berpindah
    // satu indeks setelahnya
    Node ndInput = new Node();
    if(index > 0){
        System.out.println("perbaiki logikanya!"
                + "kalau indeksnya -1 bagaimana???");
    } else if(index ==0){
        addFirst(input);
    } else{
        Node temp = head;
        for(int i =0;i < index; i++){
            temp = temp.next;
        }
        temp.next= new Node(input, temp.next);
        if(temp.next.next==null){
            tail=temp.next;
        }
    }
}
```

```
    void insertAt(int index, int input) {
        if (index < 0) {
            System.out.println("perbaiki logikanya!"+
                "kalau indeksnya -1 bagaimana");
        } else if (index == 0) {
            addFirst(input);
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1 && temp != null; i++) {
                temp = temp.next;
            }
            temp.next= new Node(input, temp.next);
            if (temp.next.next==null) {
                tail=temp.next;
            }
        }
    }
}
```

12. Pada class SLLMain, buatlah fungsi **main**, kemudian buat object dari class SingleLinkedList.

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL=new SingleLinkedList();
```

```
public class Main {
    Run | Debug
    public static void main(String[] args) {
        SingleLinkedList singLL = new SingleLinkedList();
```

13. Tambahkan Method penambahan data dan pencetakan data di setiap penambahannya agar terlihat perubahannya.

```
SingleLinkedList singLL=new SingleLinkedList();
singLL.print();
singLL.addFirst(890);
singLL.print();
singLL.addLast(760);
singLL.print();
singLL.addFirst(700);
singLL.print();
singLL.insertAfter(700, 999);
singLL.print();
singLL.insertAt(3, 833);
singLL.print();
```

```java
public class Main {
    Run | Debug
    public static void main(String[] args) {
        SingleLinkedList singLL = new SingleLinkedList();
        singLL.print();
        singLL.addFirst(input:890);
        singLL.print();
        singLL.addLast(input:760);
        singLL.print();
        singLL.addFirst(input:700);
        singLL.print();
        singLL.insertAfter(key:700, input:999);
        singLL.print();
        singLL.insertAt(index:3, input:833);
        singLL.print();
    }
}
```

2.1.1 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
Linked list kosong
Isi Linked List:        890
Isi Linked List:        890     760
Isi Linked List:        700     890     760
Isi Linked List:        700     999     890     760
Isi Linked List:        700     999     890     833     760
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Linked List Kosong
Isi Linked List :       890
Isi Linked List :       890     760
Isi Linked List :       700     890     760
Isi Linked List :       700     999     890     760
Isi Linked List :       700     999     890     833     760
```

2.1.2 Pertanyaan

1. Mengapa hasil compile kode program di baris pertama menghasilkan “Linked List Kosong”?

   Pesan “Linked List Kosong” muncul saat method print() dipanggil karena isEmpty() mengembalikan true, menandakan bahwa head dari linked list adalah null. Ini menunjukkan bahwa linked list belum berisi node apa pun, sehingga disimpulkan bahwa linked list kosong

2. Jelaskan kegunaan variable temp secara umum pada setiap method!

   temp adalah penunjuk yang digunakan untuk melacak posisi saat melakukan operasi pada linked list, seperti mencari, menambah, atau menyisipkan node pada posisi tertentu dalam struktur data linked list

3. Perhatikan class **SingleLinkedList**, pada method **insertAt** Jelaskan kegunaan kode berikut

```
if(temp.next.next==null) tail=temp.next;
```

Kode diatas berguna memeriksa apakah node baru (temp.next) adalah node terakhir dalam linked list. Jika ya, maka tail diperbarui untuk menunjuk ke node baru ini, karena node baru telah disisipkan di akhir linked list.

2.2 Modifikasi Elemen pada Single Linked List

Waktu percobaan : 30 menit

Didalam praktikum ini, kita akan mempraktekkan bagaimana mengakses elemen, mendapatkan indeks dan melakukan penghapusan data pada Single Linked List.:

2.2.1 Langkah-langkah Percobaan

1. Implementasikan method untuk mengakses data dan indeks pada linked list
2. Tambahkan method untuk mendapatkan data pada indeks tertentu pada class Single Linked List

```
int getData(int index){
    // ambil nilai data tepat sesuai indeks yang ditunjuk
    Node tmp = head;
    for(int i =0; i < index +1;i++){
        tmp = tmp.next;
    }
    return tmp.next.data;
}
```

```
public int getData(int index) {
    Node tmp = head;
    for (int i = 0; i < index +1; i++) {
        tmp = tmp.next;
    }
        return tmp.next.data;
```

3. Implementasikan method **`indexOf`**.

```
int indexOf(int key){
    // ketahui posisi nodemu ada di indeks mana
    Node tmp = head;
    int index = 0;
    while(tmp != null && tmp.data != key){
        tmp = tmp.next;
        index++;
    }
    if(tmp != null){
        return 1;
    } else{
        return index;
    }
}
```

```
int indexOf(int key){
    Node tmp = head;
    int index = 0;
    while(tmp != null && tmp.data != key){
        tmp = tmp.next;
        index++;
    }
    if(tmp == null){
        return -1;
    } else {
        return index;
    }
}
```

4. Tambahkan method removeFirst pada class SingleLinkedList

```
void removeFirst(){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else if(head == tail){
        head = tail = null;
    } else{
        head = head.next;
    }
}
```

```
void removeFirst() {
    if (isEmpty()) {
        System.out.println("Linked list masih kosong,"
            + "tidak dapat dihapus");
    } else if (head == tail) {
        head = tail = null;
    } else {
        head = head.next;
    }
}
```

5. Tambahkan method untuk menghapus data pada bagian belakang pada class SingleLinkedList

```
void removeLast(){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else if(head != tail){
        head = tail = null;
    } else{
        Node temp = head;
        while(temp.next != null){
            temp= temp.next;
        }
        temp.next = null;
        tail = temp.next;
    }
}
```

```
void removeLast() {
    if (isEmpty()) {
        System.out.println("Linked list masih kosong,"
            + "tidak dapat dihapus");
    } else if (head != tail) {
        head = tail = null;
    } else {
        Node temp = head;
        while (temp.next != null) {
            temp = temp.next;
        }
        temp.next = null;
        tail = temp.next;
    }
}
```

6. Sebagai langkah berikutnya, akan diimplementasikan method remove

```
void remove(int key){
    if(!isEmpty()){
        System.out.println("Linked list masih kosong,"
                + "tidak dapat dihapus");
    }else{
        Node temp = head;
        while(temp!=null){
            if(temp.data != key && temp==head){
                removeFirst();
                break;
            } else if(temp.next.data == key){
                temp.next = temp.next.next;
                if(temp.next == null){
                    tail = temp;
                }

                break;
            }
            temp = temp.next;
        }
    }
}
```

```
void remove (int key){
if(!isEmpty()){
    System.out.println("Linked list masih kosong,"
            + "tidak dapat dihapus");
}else{
    Node temp = head;
    while(temp!=null){
        if(temp.data != key && temp==head) {
            removeFirst();
            break;
        } else if(temp.next.data == key) {
            temp.next = temp.next.next;
            if(temp.next == null){
                tail = temp;
            }
            break;
        }
        temp = temp.next;
```

7. Implementasi method untuk menghapus node dengan menggunakan index.

```
public void removeAt(int index) {
    if (index == 0) {
        removeFirst();
    } else {
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            temp = temp.next;
        }
        temp.next = temp.next.next;
        if (temp.next == null) {
            tail = temp;
        }
    }
}
```

```
public void removeAt(int index) {
    if (index == 0) {
        removeFirst();
    } else {
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            temp = temp.next;
        }
        temp.next = temp.next.next;
        if (temp.next == null) {
            tail = temp;
        }
    }
}
```

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

```
System.out.println("Data pada indeks ke-1="+singLL.getData(1));
System.out.println("Data 3 berada pada indeks ke-"+singLL.indexOf(760));

singLL.remove(999);
singLL.print();
singLL.removeAt(0);
singLL.print();
singLL.removeFirst();
singLL.print();
singLL.removeLast();
singLL.print();
```

```
System.out.println("Data pada indeks ke -1= "+singLL.getData(index:1));
System.out.println("Data 3 berada pada indeks ke-"+singLL.indexOf(key:3));

singLL.remove(key:999);
singLL.print();
singLL.removeAt(index:0);
singLL.print();
singLL.removeFirst();
singLL.print();
singLL.removeLast();
singLL.print();
```

9. Method SLLMain menjadi:

```
public class SLLMain {
    public static void main(String[] args) {
        SingleLinkedList singLL=new SingleLinkedList();
        singLL.print();
        singLL.addFirst(890);
        singLL.print();
        singLL.addLast(760);
        singLL.print();
        singLL.addFirst(700);
        singLL.print();
        singLL.insertAfter(700, 999);
        singLL.print();
        singLL.insertAt(3, 833);
        singLL.print();

        System.out.println("Data pada indeks ke-1="+singLL.getData(1));
        System.out.println("Data 3 berada pada indeks ke-"+singLL.indexOf(760));

        singLL.remove(999);
        singLL.print();
        singLL.removeAt(0);
        singLL.print();
        singLL.removeFirst();
        singLL.print();
        singLL.removeLast();
        singLL.print();

    }
}
```

10. Jalankan class SLLMain

2.2.2 Verifikasi Hasil Percobaan

Cocokkan hasil compile kode program anda dengan gambar berikut ini.

```
run:
Linked list kosong
Isi Linked List:        890
Isi Linked List:        890     760
Isi Linked List:        700     890     760
Isi Linked List:        700     999     890     760
Isi Linked List:        700     999     890     833     760
Data pada indeks ke-1=999
Data 3 berada pada indeks ke-4
Isi Linked List:        700     890     833     760
Isi Linked List:        890     833     760
Isi Linked List:        833     760
Isi Linked List:        833
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
Linked List Kosong
Isi Linked List :       890
Isi Linked List :       890     760
Isi Linked List :       700     890     760
Isi Linked List :       700     999     890     760
Isi Linked List :       700     999     890     833     760
Data pada indeks ke -1= 999
Data 3 berada pada indeks ke -4
Isi Linked List :       700     890     833     760
Isi Linked List :       890     833     760
Isi Linked List :       833     760
Isi Linked List :       833
```

2.2.3 Pertanyaan

1. Mengapa digunakan keyword break pada fungsi remove? Jelaskan!

   digunakan untuk menghentikan loop setelah menemukan dan menghapus node yang sesuai. Ini menghindari pemeriksaan berlebihan terhadap sisa elemen dalam linked list setelah operasi penghapusan selesai

2. Jelaskan kegunaan kode dibawah pada method remove

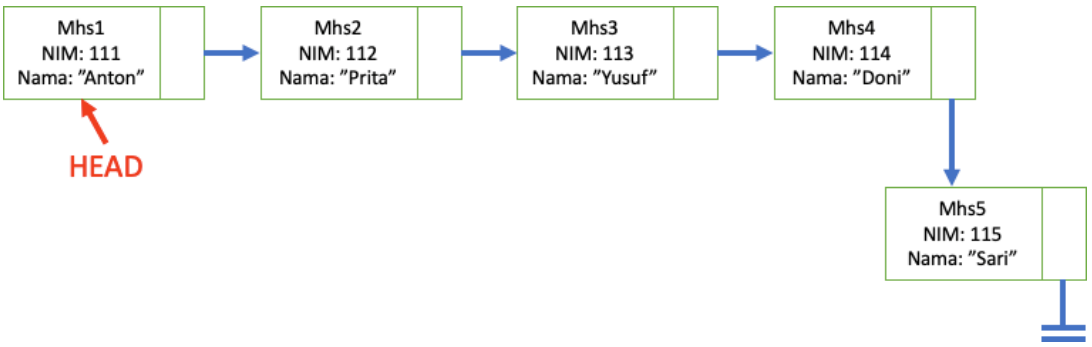
```
else if (temp.next.data == key) {
  temp.next = temp.next.next;
```

   Metode ini mengambil bilangan integer sebagai input dan menghapus node pada indeks yang sesuai dari daftar.

3. Tugas

Waktu pengerjaan : 50 menit

1 Implementasikan ilustrasi Linked List Berikut. Gunakan 4 macam penambahan data yang telah dipelajari sebelumnya untuk menginputkan data.



2 Buatlah implementasi program antrian layanan unit kemahasiswaan sesuai dengan kondisi yang ditunjukkan pada soal nomor 1! Ketentuan

a. Implementasi antrian menggunakan Queue berbasis Linked List!
b. Program merupakan proyek baru, bukan modifikasi dari soal nomor 1!

Node:

```
package Tugas;

public class Node {
    int nim;
    String nama;
    Node next;

    Node(int nim, String nama) {
        this.nim = nim;
        this.nama = nama;
        this.next = null;
    }
}
```

SingleLinkedList:

```
package Tugas;

public class SingleLinkedList {
    Node head, tail;

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

    void print() {
        if (!isEmpty()) {
            Node temp = head;
            System.out.print("Isi Linked List: ");
            while (temp != null) {
                System.out.print("\tNIM: " + temp.nim + ", Nama: " + temp.nama);
```

```
            temp = temp.next;
        }
        System.out.println();
    } else {
        System.out.println("Linked List Kosong");
    }
}

void addFirst(int nim, String nama) {
    Node newNode = new Node(nim, nama);
    if (isEmpty()) {
        head = newNode;
        tail = newNode;
    } else {
        newNode.next = head;
        head = newNode;
    }
}

void addLast(int nim, String nama) {
    Node newNode = new Node(nim, nama);
    if (isEmpty()) {
        head = newNode;
        tail = newNode;
    } else {
        tail.next = newNode;
        tail = newNode;
    }
}

void insertAfter(int keyNim, int nim, String nama) {
    Node newNode = new Node(nim, nama);
    Node temp = head;
    while (temp != null) {
        if (temp.nim == keyNim) {
            newNode.next = temp.next;
            temp.next = newNode;
            if (newNode.next == null) {
                tail = newNode;
            }
            break;
        }
        temp = temp.next;
    }
}

void insertAt(int index, int nim, String nama) {
    if (index < 0) {
        System.out.println("Indeks tidak boleh negatif");
    } else if (index == 0) {
```

```
            addFirst(nim, nama);
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1 && temp != null; i++) {
                temp = temp.next;
            }
            if (temp != null) {
                temp.next = new Node(nim, nama);
                if (temp.next.next == null) {
                    tail = temp.next;
                }
            } else {
                System.out.println("Indeks melebihi jumlah elemen dalam list");
            }
        }
    }

    int getData(int index) {
        if (index < 0) {
            System.out.println("Indeks tidak boleh negatif");
            return -1;
        }
        Node temp = head;
        for (int i = 0; i < index; i++) {
            if (temp == null) {
                System.out.println("Indeks melebihi jumlah elemen dalam list");
                return -1;
            }
            temp = temp.next;
        }
        return temp != null ? temp.nim : -1;
    }

    int indexOf(int keyNim) {
        Node temp = head;
        int index = 0;
        while (temp != null && temp.nim != keyNim) {
            temp = temp.next;
            index++;
        }
        return temp == null ? -1 : index;
    }

    void removeFirst() {
        if (isEmpty()) {
            System.out.println("Linked list masih kosong, tidak dapat dihapus");
        } else if (head == tail) {
            head = tail = null;
        } else {
            head = head.next;
```

```
        }
    }

    void removeLast() {
        if (isEmpty()) {
            System.out.println("Linked list masih kosong, tidak dapat dihapus");
        } else if (head == tail) {
            head = tail = null;
        } else {
            Node temp = head;
            while (temp.next != tail) {
                temp = temp.next;
            }
            temp.next = null;
            tail = temp;
        }
    }

    void remove(int keyNim) {
        if (isEmpty()) {
            System.out.println("Linked list masih kosong, tidak dapat dihapus");
        } else if (head.nim == keyNim) {
            removeFirst();
        } else {
            Node temp = head;
            while (temp != null) {
                if (temp.next != null && temp.next.nim == keyNim) {
                    temp.next = temp.next.next;
                    if (temp.next == null) {
                        tail = temp;
                    }
                    break;
                }
                temp = temp.next;
            }
        }
    }

    void removeAt(int index) {
        if (index < 0) {
            System.out.println("Indeks tidak boleh negatif");
        } else if (index == 0) {
            removeFirst();
        } else {
            Node temp = head;
            for (int i = 0; i < index - 1; i++) {
                if (temp == null) {
                    System.out.println("Indeks melebihi jumlah elemen dalam
list");
                    return;
```

```
                }
                temp = temp.next;
            }
            if (temp != null && temp.next != null) {
                temp.next = temp.next.next;
                if (temp.next == null) {
                    tail = temp;
                }
            } else {
                System.out.println("Indeks melebihi jumlah elemen dalam list");
            }
        }
    }
}
```

Main :

```
package Tugas;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        SingleLinkedList singLL = new SingleLinkedList();
        singLL.print();

        singLL.addLast(nim:111, nama:"Anton");
        singLL.print();

        singLL.addLast(nim:112, nama:"Prita");
        singLL.print();

        singLL.addLast(nim:113, nama:"Yusuf");
        singLL.print();

        singLL.addLast(nim:114, nama:"Doni");
        singLL.print();

        singLL.addLast(nim:115, nama:"Sari");
        singLL.print();
    }
}
```

Hasil:

```
Linked List Kosong
Isi Linked List:        NIM: 111, Nama: Anton
Isi Linked List:        NIM: 111, Nama: Anton   NIM: 112, Nama: Prita
Isi Linked List:        NIM: 111, Nama: Anton   NIM: 112, Nama: Prita   NIM: 113, Nama: Yusuf
Isi Linked List:        NIM: 111, Nama: Anton   NIM: 112, Nama: Prita   NIM: 113, Nama: Yusuf   NIM: 114, Nama: Doni
Isi Linked List:        NIM: 111, Nama: Anton   NIM: 112, Nama: Prita   NIM: 113, Nama: Yusuf   NIM: 114, Nama: Doni    NIM: 115, Nama: Sari
```