



JOBSHEET IX LINKED LIST

Nama: Rizqi Bagus Andrean

Absen: 25

Kelas: Ti-1D

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;
```

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;
}
```

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");
    }
}
```

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;
    }
}
```

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;
    }
}
```

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
}
```

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 indeks nya -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;
        }
    }
}
```

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();
    }
}
```

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();
```



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)
```

2.1.2 Pertanyaan

1. Mengapa hasil compile kode program di baris pertama menghasilkan “Linked List Kosong”?
Karena belum ada yang ditambahkan atau headnya == null
2. Jelaskan kegunaan variable temp secara umum pada setiap method!
Secara umum, variabel bernama `temp` berfungsi sebagai referensi sementara di setiap method dari class `SingleLinkedList`. Variabel ini digunakan untuk menelusuri linked list tanpa mengubah pointer `head` atau `tail` secara langsung. Ini memungkinkan method untuk iterasi melalui list, menemukan node tertentu, atau memasukkan node baru pada posisi yang diinginkan.
3. Perhatikan class **SingleLinkedList**, pada method **insertAt** Jelaskan kegunaan kode berikut

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

Potongan kode ini berfungsi untuk memperbarui pointer `tail` agar mengarah ke node setelah `temp` **tetapi hanya jika** node tersebut memang merupakan node terakhir di linked list. Situasi ini biasanya terjadi ketika kita sedang menambahkan node baru 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;
}
```



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;
    }
}
```

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;
    }
}
```

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;
    }
}
```

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;
        }
    }
}
```

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;
        }
    }
}
```

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();
```

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)
```

2.2.3 Pertanyaan

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

Break digunakan untuk menghentikan while saat satu kondisi terpenuhi;

2. Jelaskan kegunaan kode dibawah pada method remove

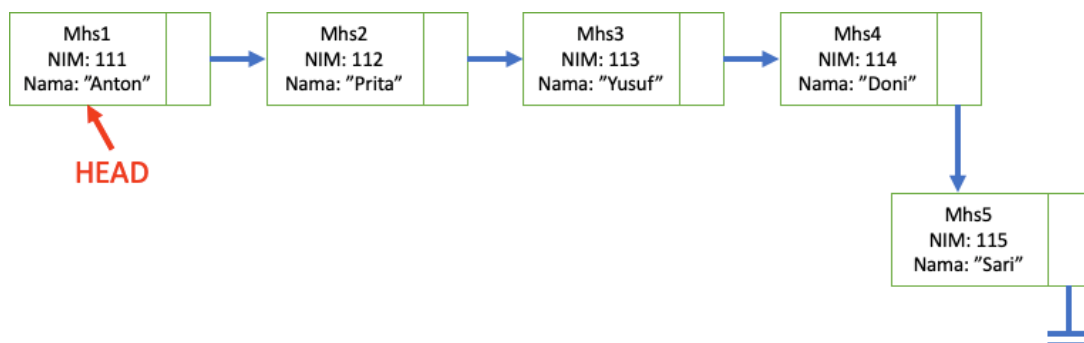
```
else if (temp.next.data == key) {
    temp.next = temp.next.next;
```

potongan kode ini melakukan bypass pada node yang ingin dihapus dengan memperbarui referensi pada node sebelumnya agar terhubung langsung ke node setelahnya. Ini secara efektif menghapus node yang ditargetkan dari linked list.

3. Tugas

Waktu pengerjaan : 50 menit

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





```
t_ws\jobsheet 10_887a23ed\bin' 'tugas1.LinkedList'
List is empty
Anton
Anton Prita
Anton Prita Yusuf
Anton Prita Yusuf Doni
Anton Prita Yusuf Doni Sari
PS C:\Users\Acer\Tugas Kuliah\Semester 2\Praktek Algoritma\jobsheet 10>
```

```
package tugas1;

public class LinkedList {

    Node head, tail;

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

    void print() {

        if (isEmpty()) {
            System.out.println("List is empty");
            return;
        }

        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data.nama + " ");
            temp = temp.next;
        }
        System.out.println();
    }

    void addFirst(Mahasiswa data) {
        Node newNode = new Node(data, null);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            newNode.next = head;
            head = newNode;
        }
    }

    void addLast(Mahasiswa data) {
        Node newNode = new Node(data, null);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            tail.next = newNode;
            tail = newNode;
        }
    }
}
```



```

    }
}

void insertAfter(Mahasiswa key, Mahasiswa input){
    Node temp = head;
    while (temp != null && temp.data != key) {
        temp = temp.next;
    }
    if (temp == null) {
        System.out.println("Key not found");
    } else {
        Node newNode = new Node(input, temp.next);
        newNode.next = temp.next;
        temp.next = newNode;
        if (temp == tail) {
            tail = newNode;
        }
    }
}

void insertAt(int index, Mahasiswa input) {
    Node ndInput = new Node();
    if (index > 0) {
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            if (temp == null) {
                System.out.println("Index out of bound");
                return;
            }
            temp = temp.next;
        }
        ndInput.data = input;
        ndInput.next = temp.next;
        temp.next = ndInput;
        if (temp == tail) {
            tail = ndInput;
        }
    } else if (index == 0) {
        ndInput.data = input;
        ndInput.next = head;
        head = ndInput;
        if (tail == null) {
            tail = ndInput;
        }
    } else {
        Node temp = head;
        for (int i = 0; i < index - 1; i++) {
            temp = temp.next;
        }
        temp.next = new Node(input, temp.next);
    }
}

```



```

        if(temp.next.next == null){
            tail = temp.next;
        }

        System.out.println("Invalid index");
    }
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();
    list.print();
    list.addFirst(new Mahasiswa("Anton", 111));
    list.print();
    list.addLast(new Mahasiswa("Prita", 124));
    list.print();
    list.addLast(new Mahasiswa("Yusuf", 124));
    list.print();
    list.addLast(new Mahasiswa("Doni", 124));
    list.print();
    list.addLast(new Mahasiswa("Sari", 124));
    list.print();
}
}

```

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!

```

● PS C:\Users\Acer\Tugas Kuliah\Semester 2\Praktek Algoritma\jobsheet
Acer\AppData\Roaming\Code\User\workspaceStorage\7e9f9bcda8eed1c74ea
Anton Masuk Antrian
Prita Masuk Antrian
Yusuf Masuk Antrian
Doni Masuk Antrian
Sari Masuk Antrian

Antrian saat ini :
Anton Prita Yusuf Doni Sari
Anton Dipanggil
Prita Dipanggil

Antrian saat ini :
Yusuf Doni Sari
○ PS C:\Users\Acer\Tugas Kuliah\Semester 2\Praktek Algoritma\jobsheet

```

```

package tugas1;

public class LinkedList {

```



```

Node head, tail;

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

void print() {

    if (isEmpty()) {
        System.out.println("List is empty");
        return;
    }

    Node temp = head;
    while (temp != null) {
        System.out.print(temp.data.nama + " ");
        temp = temp.next;
    }
    System.out.println();
}

void addLast(Mahasiswa data) {
    Node newNode = new Node(data, null);
    if (isEmpty()) {
        head = tail = newNode;
    } else {
        tail.next = newNode;
        tail = newNode;
    }
    System.out.println(data.nama + " Masuk Antrian");
}

void removeFirst() {
    if (isEmpty()) {
        System.out.println("List is empty");
        return;
    }

    System.out.println(head.data.nama + " Dipanggil");
    head = head.next;
    if (head == null) {
        tail = null;
    }
}

public static void main(String[] args) {
    LinkedList list = new LinkedList();

    // Menambah Antrian

```



```
list.addLast(new Mahasiswa("Anton", 111));
list.addLast(new Mahasiswa("Prita", 124));
list.addLast(new Mahasiswa("Yusuf", 124));
list.addLast(new Mahasiswa("Doni", 124));
list.addLast(new Mahasiswa("Sari", 124));
System.out.println();

System.out.println("Antrian saat ini : ");
list.print();

// Memanggil Antrian
list.removeFirst();
list.removeFirst();

System.out.println();
System.out.println("Antrian saat ini : ");
list.print();

}

}
```