LAPORAN PRAKTIKUM STRUKTUR DATA

LINKED LIST

OLEH:

AHMAD ZAHRAN

2411532004

DOSEN PENGAMPU:

DR. WAHYUDI, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2024

A. Pendahuluan

Struktur data Single Linked List (SLL) merupakan salah satu struktur data linier yang terdiri dari node-node yang saling terhubung satu arah. Setiap node terdiri dari dua bagian: data dan pointer ke node berikutnya. SLL digunakan secara luas karena efisien dalam penambahan dan penghapusan elemen dibandingkan array.

B. Tujuan

Tujuan dilakukannya praktikum ini adalah sebagai berikut:

1. Memahami Struktur Data LinkedList pada java
2. Mengimplementasikan LinkedList

C. Langkah Kerja Praktikum

a. Kelas NodeSLL

```
public class NodeSLL {
    int data;   // node bagian data
    NodeSLL next;   // pointer ke node berikutnya
    // konstruktor menginisialisasi node dengan data
    public NodeSLL(int data) {
        this.data = data;
        this.next = null;
    }
```

1. Setiap node menyimpan satu nilai data bertipe integer dan pointer yang menunjuk ke node berikutnya.

b. Kelas TambahSLL

```
public static NodeSLL insertAtFront(NodeSLL head, int value) {
    NodeSLL new_node = new NodeSLL(value);
    new_node.next = head;
    return new_node;
}
```

1. Menambah node di depan, fungsi ini akan menambah node baru di posisi pertama sebagai head.

```
public static NodeSLL insertAtEnd(NodeSLL head, int value) {
    // buat sebuah node dengan sebuah nilai
    NodeSLL newNode = new NodeSLL(value);

    // jika list kosong maka node jadi head
    if (head == null) {
        return newNode;
    }

    // simpan head ke variabel sementara
    NodeSLL last = head;
    // telusuri ke node akhir
    while (last.next != null) {
        last = last.next;
    }
```

2. Menambah node di akhir, dengan cara menulusuri list dari head sampai ke ode yang terakhir

```
static NodeSLL GetNode(int data) {
    return new NodeSLL(data);
}
```

3. Membuat fungsi pembantu untuk membuat node baru dengan nilai data, untuk digunakan pada method insertPos

```
static NodeSLL insertPos (NodeSLL headNode, int position, int value) {
    NodeSLL head = headNode;
    if(position < 1)
        System.out.println("Invalid position");
    if (position == 1) {
        NodeSLL new_node = new NodeSLL(value);
        new_node.next = head;
        return new_node;
    } else {
        while (position-- != 0) {
            if (position == 1) {
                NodeSLL newNode = GetNode(value);
                newNode.next = headNode.next;
                headNode.next = newNode;
                break;
            }
            headNode = headNode.next;
        }
        if (position != 1) {
            System.out.print("Posisi di luar jangkauan");
        }

        return head;
    }
```

4. Menyisipkan node di posisi tertentu.

```
public static void printList(NodeSLL head) {
    NodeSLL curr = head;
    while (curr.next != null) {
        System.out.print(curr.data + "-->");
        curr = curr.next;
    }
    if (curr.next == null) {
        System.out.print(curr.data);        }
    System.out.println();
    }
```

5. Menelusuri dan mencetak isi linked list dari depan ke belakang dan setiap nilai node dicetak diikuti oleh tanda → dan Node terakhir dicetak tanpa panah

```
public static void main(String[] args) {
    // buat linked list 2 -> 3 -> 5 -> 6
    NodeSLL head = new NodeSLL(2);
    head.next = new NodeSLL(3);
    head.next.next = new NodeSLL(5);
    head.next.next.next = new NodeSLL(6);

    // cetak list asli
    System.out.print("Senarai berantai awal:");
    printList(head);
```

6. Membuat linked List Awal

```
    // tambahkan node baru di depan
    System.out.print("Tambah 1 simpul di depan: ");
    int data = 1;
    head = insertAtFront(head, data);
    // cetak update list
    printList(head);
```

7. Menambahkan node 1 di depan

```
    // tambahkan node baru di belakang
    System.out.print("Tambah 1 simpul di belakang: ");
    int data2 = 7;
    head = insertAtEnd(head, data2);
    // cetak update list
    printList(head);
```

8. Menambahkan node 7 di belakang

```
    // tambahkan node di posisi tertentu
    System.out.print("Tambah 1 simpul ke data 4: ");
    int data3 = 4;
    int pos = 4;
    head = insertPos(head, pos, data3);
    // cetak update list
    printList(head);
    }
```

9. Menambahkan node 4 di posisi ke -4

c. Kelas HapusSLL

```
// fungsi untuk menghapus head
public static NodeSLL deleteHead(NodeSLL head) {
    // jika SLL kosong
    if (head == null)
        return null;

    // pindahkan head ke node berikutnya
    head = head.next;

    // Return head baru
    return head;
}
```

1. Menghapus Node pertama, Fungsi ini menghapus node paling awal (head). Jika list kosong (head == null), langsung kembalikan null. Jika tidak kosong, pointer head digeser ke node berikutnya. Node lama tidak dirujuk lagi sehingga akan dibersihkan otomatis oleh garbage collector.

```
public static NodeSLL removeLastNode(NodeSLL head) {
    // jika list kosong, return null
    if (head == null)
        return null;

    // jika list satu node, hapus node dan return null
    if (head.next == null)
        return null;

    // temukan node terakhir ke dua
    NodeSLL secondLast = head;
    while (secondLast.next.next != null) {
        secondLast = secondLast.next;
    }

    // hapus node terakhir
    secondLast.next = null;
    return head;
}
```

2. Menghapus Node Terakhir, buat if untuk mengecek Jika list kosong atau hanya memiliki 1 node, langsung return null. Fungsi menelusuri node sampai ke node kedua terakhir (secondLast). kemudian, pointer secondLast.next diubah menjadi null untuk menghapus node terakhir.

```
    // fungsi menghapus node di posisi tertentu
public static NodeSLL deleteNode(NodeSLL head, int position) {
    NodeSLL temp = head;
    NodeSLL prev = null;

    // jika linked list null
    if (temp == null)
        return head;

    // kasus 1: head dihapus
    if (position == 1) {
        head = temp.next;
        return head;
    }

    // kasus 2: menghapus node di tengah
    // telusuri ke node yang dihapus
    for (int i = 1; temp != null && i < position; i++) {
        prev = temp;
        temp = temp.next;
    }

    // jika ditemukan, hapus node
    if (temp != null) {
        prev.next = temp.next;
    } else {
        System.out.println("Data tidak ada");
    }

    return head;
}
```

3. Menghapus node di posisi tertentu Fungsi ini menghapus node berdasarkan posisi (berbasis 1). Jika posisi 1, artinya menghapus head. Jika tidak, lakukan perulangan sampai node sebelum posisi target (prev), dan ubah pointer-nya ke node setelah target (temp.next). Jika temp == null, berarti posisi melebihi panjang list.

```
// fungsi mencetak SLL
public static void printList(NodeSLL head) {
    NodeSLL curr = head;
    while (curr.next != null) {
        System.out.print(curr.data + "-->");
        curr = curr.next;
    }
    if (curr.next == null) {
        System.out.print(curr.data);
    }
    System.out.println();
```

4. Menelusuri dan mencetak isi linked list dari depan ke belakang dan setiap nilai node dicetak diikuti oleh tanda → dan Node terakhir dicetak tanpa panah

```
public static void main(String[] args) {
    // buat SLL 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
    NodeSLL head = new NodeSLL(1);
    head.next = new NodeSLL(2);
    head.next.next = new NodeSLL(3);
    head.next.next.next = new NodeSLL(4);
    head.next.next.next.next = new NodeSLL(5);
    head.next.next.next.next.next = new NodeSLL(6);

    // cetak list awal
    System.out.println("List awal: ");
    printList(head);

    // hapus head
    head = deleteHead(head);
    System.out.println("List setelah head dihapus: ");
    printList(head);

    // hapus node terakhir
    head = removeLastNode(head);
    System.out.println("List setelah simpul terakhir dihapus: ");
    printList(head);

    // Deleting node at position 2
    int position = 2;
    head = deleteNode(head, position);

    // Print list after deletion
    System.out.println("List setelah posisi 2 dihapus: ");
    printList(head);
    }
```

5. Kelas main yang mana proses-prosesnya ada, membuat list awal, menghapus node pertama, menghapus node treakhir, dan menghapus node di posisi ke-2

d. Kelas PencarianSLL

```
static boolean searchKey(NodeSLL head, int key) {
    NodeSLL curr = head;
    while (curr != null) {
        if (curr.data == key)
            return true;
        curr = curr.next;
    }
    return false;
}
```

1. Fungsi searchKey, digunakan untuk mencari nilai tertentu (key) dalam linkedlist.

```
public static void traversal(NodeSLL head) {
    // mulai dari head
    NodeSLL curr = head;
    // telusuri sampai pointer null
    while (curr != null) {
        System.out.print(" " + curr.data);
        curr = curr.next;
    }
    System.out.println();
}
```

2. Fungsi traversal, Fungsi ini digunakan untuk mencetak isi dari linked list. Dengan cara Menelusuri setiap node dari head hingga node terakhir (null). Mencetak setiap nilai data yang ada dalam list.

```
public static void main(String[] args) {
    NodeSLL head = new NodeSLL(14);
    head.next = new NodeSLL(21);
    head.next.next = new NodeSLL(13);
    head.next.next.next = new NodeSLL(30);
    head.next.next.next.next = new NodeSLL(10);

    System.out.print("Penelusuran SLL : ");
    traversal(head);

    // data yang akan dicari
    int key = 30;
    System.out.print("cari data " + key + " = ");
    if (searchKey(head, key))
        System.out.println("ketemu");
    else
        System.out.println("tidak ada");
}
```

3. Fungsi main, Membuat linked list secara manual, Memanggil fungsi traversal() untuk menampilkan isi list. Menentukan nilai key yang ingin dicari (dalam hal ini 30). Memanggil fungsi searchKey() untuk mencari nilai tersebut. Menampilkan hasil pencarian: ketemu jika ditemukan, tidak ada jika tidak.

D. Kesimpulan

Melalui praktikum ini, mahasiswa memperoleh pemahaman yang lebih dalam tentang bagaimana struktur data Single Linked List bekerja. Dengan membagi implementasi ke dalam beberapa kelas terpisah, program menjadi lebih modular dan mudah dikelola. Operasi penambahan, penghapusan, dan pencarian telah berhasil diimplementasikan dan diuji menggunakan Java. Pemahaman ini sangat penting untuk mengembangkan kemampuan pengelolaan data dinamis dan efisien dalam rekayasa perangkat lunak.