LAPORAN PRATIKUM STRUKTUR DATA
OPERASI SINGLY LINKED LIST (SLL)

MATA KULIAH

DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T

OLEH:

AUFAN TAUFIQURRAHMAN

NIM 2411532011

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. PENDAHULUAN

Singly Linked List (SLL) adalah struktur data linear yang terdiri dari node-node terhubung secara searah. Setiap node menyimpan data dan referensi ke node berikutnya. Praktikum ini bertujuan mengimplementasikan operasi dasar SLL: penambahan, penghapusan, dan pencarian node.

B. TUJUAN

- Memahami struktur dasar SLL dan cara pembuatan node.
- Mengimplementasikan operasi pencarian data dalam SLL.
- Menguasai teknik penambahan node di depan**,** belakang, dan posisi tertentu.
- Menguasai teknik penghapusan node di head, tail, dan posisi tertentu.
- Menganalisis kompleksitas dan error handling pada operasi SLL.

C. PROSEDUR

1. Class NodeSLL.java

```java
public class NodeSLL {
    int data;
    NodeSLL next;
    public NodeSLL(int data) {
        this.data = data;
        this.next = null;
    }
}
```

- int data: Menyimpan nilai/data pada node.
- NodeSLL next: Pointer ke node berikutnya.
- Konstruktor: Inisialisasi data dan next = null (default).

2. Class PencarialSLL,java

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

- Menerima head (node awal) dan key (data yang dicari).
- Menggunakan loop while untuk traversal dari head ke null.
- Jika key ditemukan, return true; jika tidak, return false.

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

- Mencetak semua data dalam list dengan loop while.

```
public static void main(String[] args) {
    NodeSLL head = new NodeSLL(data:14);
    head.next = new NodeSLL(data:21);
    head.next.next = new NodeSLL(data:13);
    head.next.next.next = new NodeSLL(data:30);
    head.next.next.next.next = new NodeSLL(data:10);
    System.out.print(s:"Penelusuran SLL : ");
    traversal(head);
    int key = 30;
    System.out.print("cari data " +key+ " = ");
    if (searchKey(head, key))
        System.out.println(x:"ketemu");
    else
        System.out.println(x:"tidak ada");
}
```

- Membuat SLL dengan 5 node.
- Memanggil traversal untuk mencetak list.
- Memanggil searchKey untuk mencari data 30.

Output:

```
Penelusuran SLL :  14 21 13 30 10
cari data 30 = ketemu
```

3. Class TambahSLL.java

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

- NodeSLL new_node = new NodeSLL(value) → Buat node baru.
- new_node.next = head → Hubungkan node baru ke head lama.
- return new_node → Node baru menjadi head baru.

```java
public static NodeSLL insertAtEnd(NodeSLL head, int value) {
    NodeSLL newNode = new NodeSLL(value);

    if (head == null) {
        return newNode;
    }

    NodeSLL last = head;

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

    last.next = newNode;
    return head;
}
```

- NodeSLL newNode = new NodeSLL(value) → Buat node baru
- Jika list kosong, langsung kembalikan newNode sebagai head.
- Loop hingga last mencapai node terakhir.
- last = last.next → Geser pointer ke node berikutnya.
- last.next = newNode → Hubungkan node terakhir ke newNode.

```
static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
    NodeSLL head = headNode;

    if (position < 1)
        System.out.print(s:"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(s:"Posisi di luar jangkauan"); }
        return head; }
```

- if (position < 1) → Validasi posisi minimal 1.
- Jika posisi = 1, tambahkan di depan.
- Loop untuk mencari posisi yang diinginkan.
  - newNode.next = headNode.next → Sisipkan newNode setelah headNode.
  - headNode.next = newNode → Hubungkan headNode ke newNode.
- Jika posisi tidak valid, cetak pesan error.

4. Class HapusSLL.java

```
public static NodeSLL deleteHead(NodeSLL head) {
    if (head == null)
        return null;
    head = head.next;
    return head;
}
```

- if (head == null) → Jika list kosong, return null.
- head = head.next → Geser head ke node berikutnya.

```java
public static NodeSLL removeLastNode(NodeSLL head){
    if (head == null) {
        return null;
    }
    if (head.next == null) {
        return null;
    }
    NodeSLL secondLast = head;
    while (secondLast.next.next != null) {
        secondLast = secondLast.next;
    }
    secondLast.next = null;
    return head;
}
```

- if (head.next == null) → Jika hanya ada 1 node, return null.
- Loop untuk mencari node kedua terakhir.
- secondLast.next = null → Putuskan koneksi ke tail.

```java
public static NodeSLL deleteNode(NodeSLL head, int position){
    NodeSLL temp = head;
    NodeSLL prev = null;
    if (temp == null)
        return head;
    if (position == 1) {
        head = temp.next;
        return head;
    }
    for (int i = 1; temp != null && i < position; i++) {
        prev = temp;
        temp = temp.next;
    }
    if (temp != null) {
        prev.next = temp.next;
    } else {
        System.out.println(x:"Data tidak ada");
    }
    return head;
}
```

- if (position == 1) → Jika hapus head, geser head ke node berikutnya

- Loop untuk mencari node di posisi tertentu.
  - prev = temp → Simpan node sebelumnya.
  - temp = temp.next → Geser ke node target.
- prev.next = temp.next → Hubungkan prev ke node setelah temp.

```
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();
}
```

- Loop while (curr.next != null) mencetak data hingga node sebelum terakhir.
- Node terakhir dicetak tanpa tanda --> (misal: 5)

```
public static void main(String[] args) {
    NodeSLL head = new NodeSLL(data:1);
    head.next = new NodeSLL(data:2);
    head.next.next = new NodeSLL(data:3);
    head.next.next.next = new NodeSLL(data:4);
    head.next.next.next.next = new NodeSLL(data:5);
    head.next.next.next.next.next = new NodeSLL(data:6);

    System.out.println(x:"list awal: ");
    printList(head);

    head = deleteHead(head);
    System.out.println(x:"list setelah head dihapus: ");
    printList(head);

    head = removeLastNode(head);
    System.out.println(x:"list setelah simpul terakhir di hapus: ");
    printList(head);

    int position = 2;
    head = deleteNode(head, position);
    System.out.println(x:"list setelah posisi 2 dihapus: ");
    printList(head);
}
```

- Buat list: 1 --> 2 --> 3 --> 4 --> 5 --> 6.

- Hapus head → 2 --> 3 --> 4 --> 5 --> 6.
- Hapus tail → 2 --> 3 --> 4 --> 5.
- Hapus node di posisi 2 → 2 --> 4 --> 5.

D. KESIMPULAN

Praktikum ini mengimplementasikan operasi dasar Singly Linked List (SLL) seperti penambahan, penghapusan, dan pencarian node. Implementasi manual SLL memberikan kontrol penuh atas struktur data dan hemat memori untuk operasi di depan, tetapi rentan error jika manajemen pointer tidak akurat. Sementara itu, Java Collection*s* lebih mudah digunakan untuk data dinamis meski kurang efisien dalam alokasi memori.