

LAPORAN PRAKTIKUM STRUKTUR DATA PEKAN 5  
SINGLE LINKED LIST



OLEH :  
FADIL INSANUS SIDDIK  
(2411532013)

MATA KULIAH : PRAKTIKUM STRUKTUR DATA  
DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T.

FAKULTAS TEKNOLOGI INFORMASI  
DEPARTEMEN INFORMATIKA  
UNIVERSITAS ANDALAS

PADANG, MEI 2025

## A. PENDAHULUAN

Struktur data merupakan salah satu aspek fundamental dalam dunia informatika, karena berkaitan erat dengan cara data disimpan, diakses, dan dikelola dalam proses pemrograman. Salah satu jenis struktur data yang penting dan sering digunakan adalah linked list, khususnya single linked list (SLL). Pada single linked list, elemen-elemen data disusun secara linear, di mana setiap elemen (disebut node) berisi data dan referensi menuju elemen berikutnya.

Single linked list memiliki karakteristik dinamis, yang memungkinkan penambahan dan penghapusan elemen secara efisien tanpa perlu menggeser elemen-elemen lainnya seperti pada array. Pemahaman terhadap struktur ini menjadi dasar yang penting untuk mempelajari struktur data yang lebih kompleks seperti double linked list, stack, queue, dan tree.

Pada praktikum kali ini, mahasiswa mempelajari bagaimana membangun dan mengimplementasikan single linked list menggunakan bahasa pemrograman Java. Mahasiswa juga dilatih untuk memahami konsep node, traversal (penelusuran), serta pencarian data dalam linked list. Dengan memahami implementasi dasar ini, diharapkan mahasiswa dapat membangun fondasi logika algoritmik yang lebih kuat dalam pengelolaan data secara efisien.

## B. TUJUAN PRAKTIKUM

Adapun tujuan dari praktikum ini adalah:

1. Memahami konsep dasar single linked list dan cara kerjanya.
2. Mampu mengimplementasikan node dalam single linked list menggunakan bahasa pemrograman Java.
3. Mampu membuat dan menghubungkan beberapa node sehingga membentuk satu rangkaian linked list.
4. Mampu melakukan traversal (penelusuran) elemen-elemen pada single linked list.
5. Mampu mengimplementasikan algoritma pencarian elemen (search) dalam linked list.
6. Melatih keterampilan berpikir logis dan pemecahan masalah dalam pengolahan data terstruktur.

## C. LANGKAH LANGKAH

1. TambahSLL

```
package Pekan5;
```

- Menyatakan bahwa class berada dalam package yang bernama Pekan5.

```
public class TambahSLL {
```

- Mendefinisikan class Java bernama TambahSLL.

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

- Menambahkan node di depan dari linked list.
- Membuat simpul baru.
- Simpul baru menunjuk ke head lama.
- new\_node menjadi head baru.

```
    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;  
        }  
        //ubah pointer  
        last.next = newNode;  
        return head;  
    }
```

- Menambahkan node di akhir dari linked list.
- Membuat node baru.
- Jika list kosong, node baru jadi head.
- Simpan head ke variable sementara last.
- Telusuri sampai node terakhir.
- Hubungkan node terakhir ke node baru.

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

- Shortcut untuk membuat node baru.

```

static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
    NodeSLL head = headNode;
    if (position < 1)
        System.out.print("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;
    }
}

```

- Menambahkan node di posisi tertentu.

```
NodeSLL head = headNode;
```

- Buat Salinan pointer head untuk iterasi.

```

if (position < 1)
    System.out.print("Invalid position");

```

- Validasi agar posisi tidak negative.

```

if (position == 1) {
    NodeSLL new_node = new NodeSLL(value);
    new_node.next = head;
    return new_node;
}

```

- Jika posisi 1, insert di awal.

```

} else {
    while (position-- != 0) {
        if (position == 1) {
            NodeSLL newNode = GetNode(value);
            newNode.next = headNode.next;
            headNode.next = newNode;
            break;
        }
    }
}

```

- Siapkan node di posisi tertentu.
- Menyisipkan node setelah node saat ini.

```

    }
    if (position != 1)
        System.out.print("Posisi di luar jangkauan");
    return head;
}

```

- Jika gagal sisip, tampilkan pesan error.

```

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();
}

```

- Menampilkan semua elemen linked list.
- Tampilkan semua elemen kecuali terakhir.
- Cetak elemen terakhir tanpa -->.

```

public static void main (String[] args) {

```

- Main pada class.

```

// Node Linked List 2_7_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

```

- Membuat linked list awal: 2→3→5→6

```

System.out.print("Senarai berantai awal: ");
printList(head);

```

- Menampilkan isi list sebelum ada perubahan.

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

- Tambah node 1 di depan → 1→2→3→5→6

```
int data2 = 7;
head = insertAtEnd(head, data2);
//cetak update list
printList(head);
```

- Tambah node 7 di akhir → 1→2→3→5→6→7

```
System.out.print("tamabah 1 simpul ke data 4: ");
int data3 = 4;
int pos = 4;
head = insertPos(head, pos, data3);
//cetak update list
printList(head);
```

- Tambah 4 pada posisi ke-4 → 1→2→3→4→5→6→7
- Output:

```
<terminated> tambahSLL [Java Application] C:\Users\LENOVO\.p2\pool\plugins\c
Senarai berantai awal: 2-->3-->5-->6
tambah 1 simpul di depan: 1-->2-->3-->5-->6
tambah 1 simpul di belakang: 1-->2-->3-->5-->6-->7
tamabah 1 simpul ke data 4: 1-->2-->3-->4-->5-->6-->7
```

## 2. HapusSLL

```
public class HapusSLL {
```

- Deklarasi HapusSLL.

```
public static NodeSLL deletedHead(NodeSLL head) {
```

- Fungsi static untuk menghapus node pertama (head) dari linked list.

```
// jika SLL kosong  
if (head == null)
```

- Jika list kosong (tidak ada node), maka tidak perlu dihapus.

```
if (head == null)
```

- Return null karena tidak ada yang bisa dihapus.

```
head = head.next;
```

- Head dipindah ke node berikutnya.

```
return head; }
```

- Kembalikan head yang baru.

```
public static NodeSLL removeLastNode(NodeSLL head) {
```

- Fungsi untuk menghapus node terakhir pada linked list.

```
//jika list kosong, return null  
if (head == null) {  
    return null;  
}
```

- Jika list kosong, kembalikan null.

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

- Jika hanya ada satu node, hapus dan kembalikan null.

```
NodeSLL secondLast = head;
```

- Pointer untuk mencari node sebelum terakhir.

```
while (secondLast.next.next != null) {  
    secondLast = secondLast.next;  
}
```

- Looping sampai menemukan node ke-dua terakhir.

```
//hapus node terakhir
secondLast.next = null;
```

- Putuskan hubungan dengan node terakhir.

```
return head;
}
```

- Kembalikan head asli (karena node pertama tidak berubah).

```
public static NodeSLL deleteNode(NodeSLL head, int position) {
```

- Fungsi untuk menghapus node pada posisi tertentu.

```
NodeSLL temp = head;
NodeSLL prev = null;
```

- temp : untuk traversal.
- prev : menyimpan node sebelumnya.

```
if (temp == null)
    return head;
```

- Jika list kosong, kembalikan head.

```
if (position == 1) {
    head = temp.next;
    return head;}
}
```

- Jika posisi yang dihapus adalah posisi pertama, pindahkan head.

```
//telusuri ke node yang dihapus
for (int i = 1; temp != null && i < position; i++) {
    prev = temp;
    temp = temp.next;}
//jika ditemukan, hapus node
```

- Traversal menuju posisi yang akan dihapus.

```
if (temp != null) {
    prev.next = temp.next;
```



- Jika node ditemukan, hapus dengan melewati node tersebut.

```
}else {
    System.out.print("Data tidak ada");}
```

- Jika posisi melebihi panjang list, cetak pesan.

```
return head;}
```

- Kembalikan head.

```
public static void printList(NodeSLL head) {
```

- Fungsi untuk mencetak isi linked list.

```
NodeSLL curr = head;
```

- Pointer untuk traversal.

```
while (curr.next != null) {
    System.out.print(curr.data+"-->");
    curr = curr.next;
```

- Cetak setiap node kecuali yang terakhir dengan panah.

```
if (curr.next==null)
    System.out.print(curr.data);
System.out.println();}
}
```

- Cetak data node terakhir tanpa panah, lalu ganti baris.

```
//keals main
public static void main (String[] args) {
```

- Fungsi utama program Java.

```
//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);
```

- Membuat linked list: 1 -> 2 -> 3 -> 4 -> 5 -> 6

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

- Cetak list awal.

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

- Hapus node pertama (1) → list jadi 2 -> 3 -> 4 -> 5 -> 6.

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

- Hapus node terakhir (6) → list jadi 2 -> 3 -> 4 -> 5.

```
//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);
```

- Hapus node posisi ke-2 (3) → list jadi 2 -> 4 -> 5.
- Output :

list awal:

1-->2-->3-->4-->5-->6

List setelah head dihapus:

2-->3-->4-->5-->6

List setelah simpul terakhir dihapus:

2-->3-->4-->5

List setelah posisi 2 dihapus:

2-->4-->5

### 3. PencarianSLL

```
public class PencarianSLL {
```

- Mendeklarasikan kelas Java bernama PencarianSLL.
- Kelas ini akan berisi method untuk pencarian dan traversal (penelusuran) pada Single Linked List (SLL).

```
static boolean searchKey(NodeSLL head, int key) {
```

- Method searchKey() berfungsi untuk mencari nilai (key) dalam linked list.
- Parameter:
  - head : pointer ke node pertama.
  - Key : nilai yang ingin dicari.
- Statis : method ini bisa dipanggil tanpa membuat objek PencarianSLL.
- Boolean : return value-nya true jika ditemukan, false jika tidak.

```
NodeSLL curr = head;
```

- Buat variabel curr untuk menunjuk node saat ini, dimulai dari head.

```
while (curr != null) {
    if (curr.data == key)
        return true;
    curr = curr.next;}
}
```

- Loop selama curr belum null.
- Jika data pada node sama dengan key, kembalikan true.
- Jika belum cocok, lanjut ke node berikutnya.

```
return false;}
```

- Jika sudah sampai akhir list dan tidak ditemukan, maka kembalikan false.

```
public static void traversal (NodeSLL head) {
```

- Method untuk menelusuri dan mencetak seluruh node di list.

```
NodeSLL curr = head;
```

- Sama seperti sebelumnya, inisialisasi pointer curr untuk iterasi.

```
//telusuri sampai pointer null
while (curr != null) {
    System.out.print(" " + curr.data);
    curr = curr.next; }
```

- Loop menelusuri list.
- Cetak nilai data dari setiap node, dipisahkan spasi.

```
System.out.println(); }
```

- Tambah baris baru setelah semua elemen dicetak.

```
public static void main(String[] args) {
```

- Titik awal eksekusi program.

```
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);
```

- Membuat linked list secara manual.

```
System.out.print("Penelusuran SLL : ");
```

- Cetak teks pembuka sebelum list ditampilkan.

```
traversal(head);
```

- Panggil fungsi traversal() untuk mencetak isi list.

```
int key = 30;
```

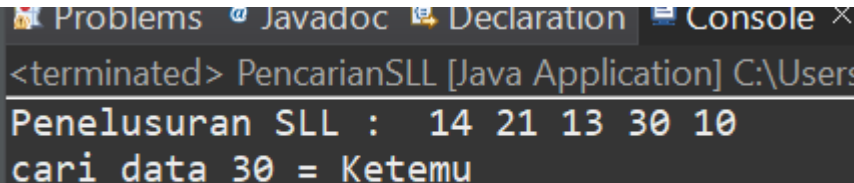
- Menyimpan nilai yang akan dicari di list ke dalam variabel key.

```
System.out.print("cari data " +key+ " = ");
```

- Mencetak teks untuk menunjukkan data yang sedang dicari.

```
if (searchKey(head, key))
    System.out.println("Ketemu");
else
    System.out.println("Tidak ada");
```

- Mengecek apakah key ditemukan di list.
- Jika ya, tampilkan "Ketemu".
- Jika tidak, tampilkan "Tidak ada".
- Output :



```
<terminated> PencarianSLL [Java Application] C:\Users...
Penelusuran SLL : 14 21 13 30 10
cari data 30 = Ketemu
```

#### 4. NodeSLL

```
3 public class NodeSLL {
```

- Mendeklarasikan kelas NodeSLL.
- Kelas ini digunakan untuk membentuk node dalam struktur data Single Linked List (SLL).

```
    int data;
```

- Variabel data menyimpan nilai dari node (dalam bentuk bilangan bulat).

```
    NodeSLL next;
```

- Variabel next adalah referensi (pointer) ke node berikutnya dalam linked list.

```
    public NodeSLL(int data)
```

- Konstruktor kelas NodeSLL, dipanggil saat membuat objek node baru.
- Parameter data digunakan untuk mengisi nilai node saat pertama kali dibuat.

```
{  
    this.data = data;  
    this.next = null;
```

- `this.data = data;` : menyimpan nilai parameter data ke variable instance data.
- `This.next = null;` : menginisialisasi pointer next dengan null, artinya belum ada node setelahnya.
- Output : Tidak ada output karena kode itu hanya mendefinisikan kelas NodeSLL, tanpa ada fungsi `main()` atau perintah eksekusi.