LAPORAN PRAKTIKUM STRUKTUR DATA

LINKED LIST

OLEH :

JOVANTRI IMMANUEL GULO

NIM 2411532014

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU :

Dr. Ir. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, 20 MEI 2025

A. Pendahuluan

Linked list (senarai berantai) merupakan salah satu struktur data yang digunakan untuk menyimpan dan mengelola data secara dinamis. Berbeda dengan array, linked list tidak memiliki ukuran tetap, sehingga memungkinkan penambahan dan penghapusan elemen dengan lebih fleksibel. Praktikum ini bertujuan untuk memahami konsep dasar linked list serta mengimplementasikan berbagai operasi yang dapat dilakukan, seperti menambahkan, menghapus, dan mencari elemen dalam linked list.

B. Tujuan

Tujuan dari dilakukannya praktikum ini adalah

1. Mampu memahami cara mengimplementasikan linked list.
2. Mampu menggunakan konstruktor dan method pada linked list.
3. Mampu mengimplementasikan linked list dalam kehidupan sehari-hari.

C. Langkah – langkah Pengerjaan

Berikut adalah langkah-langkah dalam pengerjaan praktikum kali ini:

1. Pertama, buat file class dengan nama **NodeSLL**.
2. Deklarasi data dengan tipe integer, serta variabel next yang berfungsi sebagai pointer untuk menyambungkan node berikutnya dalam linked list.

```
4       int data;
5       NodeSLL next;
```

3. Konstruktor menginisialisasi node dengan data yang diberikan, lalu next diatur ke null karena saat pertama dibuat, node belum terhubung ke yang lain.

```
7       public NodeSLL(int data) {
8           this.data = data;
9           this.next = null;
10      }
```

4. Kedua, buat file class dengan nama **HapusSLL**.
5. Deklarasi class untuk operasi penghapusan pada single linked list. Menghapus simpul pertama dengan memindahkan referensi head ke simpul berikutnya. Jika head kosong, maka mengembalikan null.

```
3  public class HapusSLL {
4      public static NodeSLL deleteHead(NodeSLL head) {
5          if(head == null)
6              return null;
7          head = head.next;
8          return head;
9      }
```

6. Menghapus simpul terakhir dengan mencari simpul kedua terakhir dan mengatur referensi next-nya menjadi null. Jika list kosong atau hanya memiliki satu elemen, mengembalikan null.

```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;
    }
```

7. Menghapus simpul pada posisi tertentu dalam linked list. Jika posisi adalah satu, maka head langsung diperbarui. Jika posisi ditemukan, simpul sebelumnya diatur agar menunjuk ke simpul setelahnya, sehingga elemen dihapus dari list. Jika posisi tidak ditemukan, ditampilkan pesan bahwa data tidak ada.

```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("Data tidak ada.");
        }
        return head;
    }
```

8. Mencetak elemen-elemen dari linked list dengan format yang menunjukkan hubungan antar simpul. Jika sudah mencapai simpul terakhir, tidak menambahkan panah setelahnya.

```java
    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();
    }
```

9. Membuat linked list berisi enam simpul, lalu melakukan penghapusan pada beberapa posisi dan mencetak hasilnya. Awalnya, list ditampilkan dalam keadaan lengkap, kemudian dilakukan penghapusan head, penghapusan simpul terakhir,

serta penghapusan simpul pada posisi tertentu. Hasil dari tiap operasi dicetak untuk memantau perubahan dalam linked list.

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

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

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

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

        int position = 2;
        head = deleteNode(head, position);

        System.out.println("List setelah posisi 2 dihapus: ");
        printList(head);
    }
```

10. Berikut adalah output dari program.

```
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
```

11. Ketiga, buat file class dengan nama **PencarianSLL**.
12. Deklarasi class untuk melakukan pencarian data dalam single linked list.
13. Fungsi searchKey mencari data tertentu dalam linked list dengan melakukan traversal dari head hingga akhir. Jika data ditemukan, mengembalikan true, jika tidak ditemukan, mengembalikan false.

```
public class 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;
    }
```

14. Fungsi traversal digunakan untuk menampilkan semua elemen dalam linked list dengan melakukan iterasi dari head hingga simpul terakhir.

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

15. Bagian main membuat linked list dengan lima elemen, lalu mencetaknya dengan traversal. Kemudian, pencarian dilakukan terhadap nilai tertentu dan hasilnya ditampilkan, apakah data ditemukan atau tidak.

```
    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);
        int key = 30; // Query yang akan dicari
        System.out.print("Cari data " + key + " = ");
        if(searchKey(head, key))
            System.out.println("Ketemu!");
        else
            System.out.println("Tidak ada.");
    }
}
```

Kode ini memastikan pencarian dalam linked list dilakukan secara berurutan, dengan kondisi berhenti jika data ditemukan atau simpul terakhir telah dilalui.

16. Berikut adalah output atau hasil dari program.

```
Penelusuran SLL:  14 21 13 30 10
Cari data 30 = Ketemu!
```

17. Keempat, buat file class dengan nama **TambahSLL**.
18. Deklarasi class untuk operasi penambahan simpul dalam single linked list.
19. Menambahkan simpul baru di awal linked list dengan mengatur referensi next dari simpul baru ke head yang lama, lalu head diperbarui agar menunjuk ke simpul baru.

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

20. Menambahkan simpul baru di akhir linked list dengan mencari simpul terakhir, kemudian mengatur referensi next dari simpul terakhir agar menunjuk ke simpul baru. Jika list kosong, simpul baru langsung dijadikan head.

```
    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;
    }
```

21. Membuat dan mengembalikan simpul baru dengan nilai data yang diberikan.

```
22      static NodeSLL GetNode(int data) {
23          return new NodeSLL(data);
24      }
```

22. Menambahkan simpul baru di posisi tertentu dalam linked list. Jika posisi adalah satu, maka simpul baru langsung menjadi head. Jika posisi ditemukan dalam list, simpul baru dimasukkan di antara dua simpul yang sudah ada dengan memperbarui referensi next. Jika posisi tidak valid, ditampilkan pesan kesalahan.

```
26      static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
27          NodeSLL head = headNode;
28          if(position < 1)
29              System.out.println("Invalid position.");
30          if(position == 1) {
31              NodeSLL new_node = new NodeSLL(value);
32              new_node.next = head;
33              return new_node;
34          } else {
35              while(position-- != 0) {
36                  if(position == 1) {
37                      NodeSLL newNode = GetNode(value);
38                      newNode.next = headNode.next;
39                      headNode.next = newNode;
40                      break;
41                  }
42                  headNode = headNode.next;
43              }
44              if(position != 1)
45                  System.out.println("Posisi di luar jangkauan.");
46          }
47          return head;
48      }
```

23. Mencetak semua elemen dalam linked list dengan format yang memperlihatkan hubungan antar simpul. Jika sudah mencapai simpul terakhir, tidak menambahkan panah setelahnya.

```
49      public static void printList(NodeSLL head) {
50          NodeSLL curr = head;
51          while(curr.next != null) {
52              System.out.print(curr.data + " ⟶ ");
53              curr = curr.next;
54          }
55          if(curr.next == null)
56              System.out.println(curr.data);
57          System.out.println();
58      }
```

24. Membuat linked list dengan empat elemen, lalu melakukan tiga operasi penambahan: satu simpul di depan, satu di belakang, dan satu pada posisi keempat. Setelah setiap operasi, list dicetak untuk memantau perubahan yang terjadi.

```
59      public static void main(String[] args) {
60          NodeSLL head = new NodeSLL(2);
61          head.next = new NodeSLL(3);
62          head.next.next = new NodeSLL(5);
63          head.next.next.next = new NodeSLL(6);
64          System.out.print("Senarai berantai awal: ");
65          printList(head);
66          System.out.print("Tambah 1 simpul di depan: ");
67          int data = 1;
68          head = insertAtFront(head, data);
69          printList(head);
70          System.out.print("Tambah 1 simpul di belakang: ");
71          int data2 = 7;
72          head = insertAtEnd(head, data2);
73          printList(head);
74          System.out.print("Tambah 1 simpul di data ke 4: ");
75          int data3 = 4;
76          int pos = 4;
77          head = insertPos(head, pos, data3);
78          printList(head);
79      }
```

25. Berikut adalah output atau keluaran dari program ini.

```
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

Tambah 1 simpul di data ke 4: 1 --> 2 --> 3 --> 4 --> 5 --> 6 --> 7
```

D. Kesimpulan

Dari praktikum yang telah dilakukan, dapat diambil kesimpulan bahwa linked list memiliki keunggulan dalam manajemen memori dibandingkan array karena strukturnya yang dinamis. Selain itu, implementasi linked list memungkinkan manipulasi data dengan efisien tanpa perlu menggeser elemen lain seperti pada array. Namun, penggunaan linked list juga memiliki tantangan, terutama dalam aspek akses data yang lebih kompleks dibandingkan struktur data lain.