LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 5: LINKED LIST (SLL)

Oleh :

SEPTIAN RIYANDA PUTRA

NIM 2411532016

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU : DR. WAHYUDI, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI
DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

A. Pendahuluan

Linked List adalah salah satu jenis struktur data linear yang terdiri dari sekumpulan elemen yang disebut node, di mana setiap node saling terhubung satu sama lain melalui *pointer* atau referensi. Berbeda dengan array yang memiliki ukuran tetap dan menyimpan elemen dalam lokasi memori yang bersebelahan, linked list menyimpan elemen di lokasi memori yang acak, dan hubungan antar elemen dijaga melalui pointer.

Pada praktikum ini, fokus utama adalah implementasi dari Single Li nked List (SLL) menggunakan bahasa pemrograman Java. Single Linked List adalah jenis linked list yang setiap nodenya hanya memiliki satu referensi menuju node berikutnya, menjadikannya struktur yang sederhana namun sangat efektif. Struktur ini umum digunakan dalam berbagai skenario, seperti pengelolaan antrian, traversing data, hingga operasi penyisipan dan penghapusan elemen secara dinamis.

Melalui kegiatan praktikum ini, kita diharapkan mampu memahami konsep dasar dan mekanisme kerja Single Linked List, mulai dari pembuatan node, penambahan elemen di akhir list, penghapusan dari awal, hingga pencarian data berdasarkan kriteria tertentu. Selain memberikan pemahaman teoretis, praktikum ini juga menekankan pada penerapan nyata di Java menggunakan cara pandang berorientasi objek melalui penggunaan kelas dan objek. Dengan begitu kita tidak hanya memahami bagaimana linked list bekerja, tetapi juga mampu mengimplementasikannya dalam pengembangan perangkat lunak secara praktis.

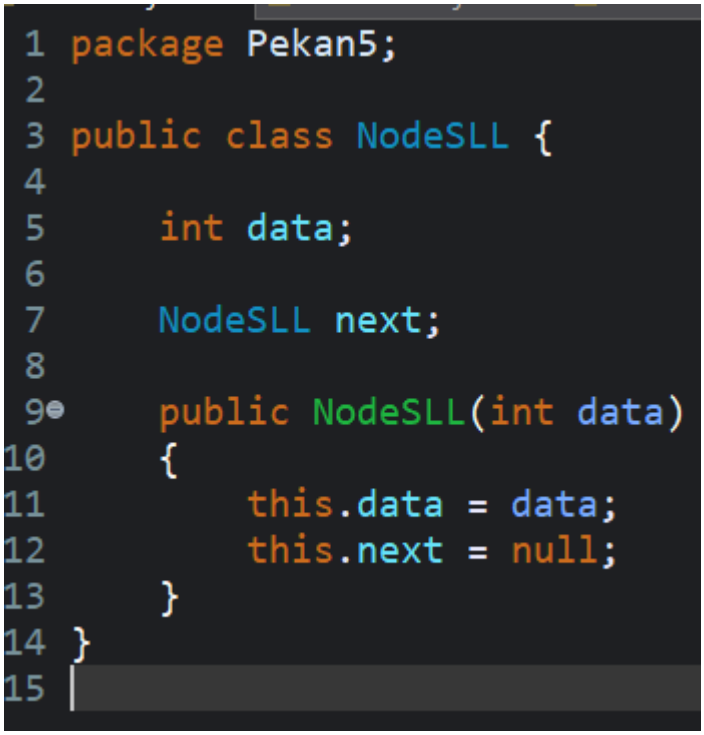
B. Tujuan

Tujuan dari dilakukannya praktikum ini adalah :

1. Mempelajari dan memahami konsep dasar struktur data *linked list*, khususnya *single linked list*.
2. Mengetahui cara menerapkan single linked list dalam praktik menggunakan bahasa pemrograman Java.
3. Memiliki kemampuan untuk membuat serta memodifikasi node pada single linked list, mencakup operasi penambahan, penghapusan, dan penelusuran data.

C. Program yang dibuat

a. NodeSLL



```java
package Pekan5;

public class NodeSLL {

    int data;

    NodeSLL next;

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

- **Data (isi node)** — dalam hal ini berupa tipe `int`.
- **Pointer ke node berikutnya** — yaitu variabel `next` yang menunjuk ke objek `NodeSLL` lain.

Konstruktor `public NodeSLL(int data)` digunakan untuk:

- Menyimpan nilai yang diberikan ke dalam atribut `data`.
- Mengatur `next` menjadi `null` sebagai default, karena saat pertama kali dibuat, node ini belum terhubung ke node lain.

b. TambahSLL

```java
package Pekan5;

public class TambahSLL {

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

Fungsi: Menambahkan node baru di depan linked list.

- Buat node baru dengan nilai `value`.
- Set `next` node baru ke `head` lama.
- Return node baru sebagai 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;
    }
```

Fungsi: Menambahkan node baru di akhir linked list.

- Jika list kosong, return node baru sebagai head.
- Jika tidak kosong, iterasi sampai ke node terakhir (`last.next == null`).
- Set `last.next` ke node baru.

```java

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

- Fungsi bantu untuk membuat node baru.
- Fungsinya sama seperti `new NodeSLL(data)`.

```
27
28    static NodeSLL insertPos(NodeSLL headNode, int position, int value) {
29        NodeSLL head = headNode;
30        if (position < 1) {
31            System.out.println("Invalid Position");
32            return head;
33        }
34
35        // posisi pertama (insert di depan)
36        if (position == 1) {
37            NodeSLL new_node = new NodeSLL(value);
38            new_node.next = head;
39            return new_node;
40        } else {
41            NodeSLL current = head;
42            while (position-- > 2) {
43                if (current == null) {
44                    System.out.println("Posisi di luar jangkauan");
45                    return head;
46                }
47                current = current.next;
48            }
49            if (current != null) {
50                NodeSLL newNode = new NodeSLL(value);
51                newNode.next = current.next;
52                current.next = newNode;
53            } else {
54                System.out.println("Posisi di luar jangkauan");
55            }
56        }
57        return head;
58    }
```

Fungsi: Menambahkan node pada posisi tertentu dalam list.

- Jika posisi 1, tambahkan di depan.
- Jika posisi > 1, iterasi hingga ke posisi-1.
- Sisipkan node baru di antara node sebelumnya dan node berikutnya.

```
59
60    public static void printList(NodeSLL head) {
61        NodeSLL curr = head;
62        while (curr != null) {
63            System.out.print(curr.data);
64            if (curr.next != null) System.out.print("-->");
65            curr = curr.next;
66        }
67        System.out.println();
68    }
```

Fungsi: Menelusuri seluruh list dan mencetak isi data dari setiap node.

- Mulai dari `head`, cetak `data`, lalu lanjut ke `next` sampai `null`.
- Gunakan format ”−−>″ untuk tampilan yang rapi.

```
69
70      public static void main(String[] args) {
71          NodeSLL head = new NodeSLL(2);
72          head.next = new NodeSLL(3);
73          head.next.next = new NodeSLL(5);
74          head.next.next.next = new NodeSLL(6);
75
76          System.out.print("Senarai berantai awal: ");
77          printList(head);
78
79          System.out.print("Tambah 1 simpul di depan: ");
80          int data = 1;
81          head = insertAtFront(head, data);
82          printList(head);
83
84          System.out.print("Tambah 1 simpul di belakang: ");
85          int data2 = 7;
86          head = insertAtEnd(head, data2);
87          printList(head);
88
89          System.out.print("Tambah 1 simpul ke posisi 4: ");
90          int data3 = 4;
91          int pos = 4;
92          head = insertPos(head, pos, data3);
93          printList(head);
94      }
95  }
```

- Digunakan untuk menjalankan program dan menguji operasi linked list.

1. **Membuat list awal**: `2 -> 3 -> 5 -> 6`
2. **Tambah di depan** (`1`) → `1 -> 2 -> 3 -> 5 -> 6`
3. **Tambah di belakang** (`7`) → `1 -> 2 -> 3 -> 5 -> 6 -> 7`
4. **Tambah di posisi ke-4** (`4`) → `1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7`

Output akan seperti berikut:

```
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 ke posisi 4: 1-->2-->3-->4-->5-->6-->7
```

c. PencarianSLL

```
1  package Pekan5;
2
3  public class PencarianSLL {
4      static boolean searchKey(NodeSLL head, int key) {
5          NodeSLL curr = head;
6          while (curr != null) {
7              if (curr.data == key)
8              return true;
9              curr = curr.next; }
10         return false; }
```

Fungsi: Mencari apakah suatu nilai (key) ada di dalam linked list.

- Mulai dari `head`
- Selama node saat ini (`curr`) tidak `null`:
    - Jika data node sama dengan `key`, return `true`
    - Jika tidak, lanjut ke node berikutnya
- Jika sudah mencapai akhir list (`null`) dan belum ditemukan, return `false`

```
11         public static void traversal (NodeSLL head) {
12             NodeSLL curr = head;
13             while(curr != null) {
14                 System.out.println(" " + curr.data);
15                 curr = curr.next; }
16                 System.out.println(); }
```

- Menelusuri seluruh node dari head hingga akhir (`curr != null`).
- Menampilkan nilai `data` dari setiap node satu per satu.

```
17          public static void main(String[] args) {
18              NodeSLL head = new NodeSLL(14);
19              head.next = new NodeSLL(21);
20              head.next.next = new NodeSLL(13);
21              head.next.next.next = new NodeSLL(30);
22              head.next.next.next.next = new NodeSLL(10);
23              System.out.println("Penelusuran SLL : ");
24              traversal(head);
25              int key = 30;
26              System.out.println("cari data " +key+ " = ");
27              if (searchKey(head, key))
28                  System.out.println("ketemu");
29              else
30                  System.out.println("tidak ada");
31          }
32      }
33
```

Fungsi: Program utama untuk menjalankan fungsi penelusuran dan pencarian.

- Membuat linked list:
  `14 -> 21 -> 13 -> 30 -> 10`
- Menampilkan isi list melalui fungsi `traversal()`.
- Mencari apakah angka `30` ada di dalam list.
- Cetak hasil pencarian.

Output akan seperti berikut:

```
Penelusuran SLL :
 14
 21
 13
 30
 10

cari data 30 =
ketemu
```

d. HapusSLL

```java
package Pekan5;

public class HapusSLL {
        public static NodeSLL deleteHead(NodeSLL head) {
            if (head == null)
                return null;

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

Fungsi: Menghapus simpul pertama (head) dari linked list.

- Jika list kosong (`head == null`), kembalikan `null`.
- Kalau tidak, pindahkan `head` ke node berikutnya (`head.next`) dan return sebagai `head` baru.

```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;
        }
```

Fungsi: Menghapus node terakhir dalam linked list.

- Jika list kosong: return `null`
- Jika hanya 1 node: return `null` (karena node tersebut dihapus)
- Kalau lebih dari 1 node: cari node **kedua terakhir**, lalu ubah `next`-nya menjadi `null`.

```java
public static NodeSLL deleteNode(NodeSLL head, int position) {
    NodeSLL temp = head;
    NodeSLL prev = null;

    if (temp == null)
        return null;

    if (position == 0) {
        head = head.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;
}
```

Fungsi: Menghapus simpul pada **posisi tertentu** (berbasis indeks ke-0).

- Jika posisi adalah `0`, artinya hapus head.
- Jika posisi di tengah atau akhir, telusuri sampai indeks tersebut, lalu hapus simpul dengan mengubah referensi `next` dari simpul sebelumnya.

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

    if (curr == null)
        System.out.print("null");

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

- Menampilkan isi dari linked list dari head hingga akhir (`null`).

```java
    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);
    }
}
```

Digunakan untuk menjalankan program dan menguji operasi linked list

- Membuat linked list: `1 → 2 → 3 → 4 → 5 → 6 → null`
- Menghapus head → `2 → 3 → 4 → 5 → 6 → null`
- Menghapus simpul terakhir → `2 → 3 → 4 → 5 → null`
- Menghapus simpul di posisi ke-2 (data 4) → `2 → 3 → 5 → null`

Output akan seperti berikut:

```
List awal:
1-->2-->3-->4-->5-->6-->null
List setelah head dihapus:
2-->3-->4-->5-->6-->null
List setelah simpul terakhir dihapus:
2-->3-->4-->5-->null
List setelah posisi 2 dihapus:
2-->4-->5-->null
```

D. Kesimpulan

Berdasarkan praktikum yang telah dilaksanakan, dapat disimpulkan bahwa *Single Linked List* (SLL) merupakan salah satu bentuk struktur data yang terdiri atas sekumpulan simpul (*node*) yang saling terhubung. Setiap simpul dalam SLL memuat dua komponen penting, yaitu data yang menyimpan nilai atau informasi, dan referensi (*pointer*) yang menunjuk ke simpul berikutnya dalam daftar. Tidak seperti array yang memiliki ukuran tetap dan lokasi elemen yang saling berdekatan di memori, linked list bersifat dinamis dan memungkinkan perubahan ukuran serta lokasi elemen tanpa perlu menggeser elemen lain.

Dalam konteks implementasi menggunakan bahasa pemrograman Java, struktur SLL dibangun melalui definisi kelas NodeSLL, yang berperan sebagai cetak biru bagi setiap simpul. Kelas ini mencakup atribut untuk menyimpan data serta referensi ke simpul berikutnya, memungkinkan pembentukan rangkaian node secara efisien di memori.

Selama praktikum, juga telah dilakukan percobaan terhadap berbagai operasi dasar pada SLL. Operasi-operasi tersebut mencakup penambahan simpul di awal, di akhir, maupun di posisi tertentu dalam senarai; penelusuran (*traversal*) untuk menampilkan isi daftar; pencarian nilai tertentu (*searching*); serta penghapusan simpul dari awal (head), akhir, dan posisi tertentu. Setiap operasi ini memberikan wawasan mendalam mengenai bagaimana manipulasi simpul dilakukan melalui pengaturan pointer, dan bagaimana perubahan pada satu simpul dapat memengaruhi struktur keseluruhan.

Dengan memahami dan mengimplementasikan berbagai operasi tersebut, kita diharapkan mampu memperoleh pemahaman yang lebih komprehensif tentang prinsip kerja linked list. Praktikum ini juga mempertegas pentingnya pengelolaan pointer dalam menjaga konsistensi alur data. Secara keseluruhan, kegiatan ini memberikan dasar yang kuat untuk memahami konsep struktur data dinamis dan relevansinya dalam pengembangan program yang efisien dan adaptif.