LAPORAN PRAKTIKUM STRUKTUR DATA

PEKAN 5: Manipulasi dan Pengelolaan Data Dengan Singly Linked List Dalam Java

OLEH

Abdullah Al Ramadhani (2411533016)

DOSEN PENGAMPU

DR. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. Pendahuluan

Struktur data menjadi salah satu komponen penting dalam pengembangan aplikasi komputer. Salah satu struktur data yang cukup sering digunakan dalam dunia pemrograman adalah *Linked List* atau daftar berantai. Dari berbagai jenis *Linked List*, *Singly Linked List* (SLL) merupakan bentuk yang paling dasar dan banyak dimanfaatkan dalam penerapan algoritma maupun pembuatan aplikasi. Struktur ini tersusun atas elemen-elemen yang disebut node, di mana setiap node menyimpan nilai data dan sebuah referensi menuju node berikutnya.

Dalam pembahasan ini, akan diulas implementasi beberapa operasi dasar pada *Singly Linked List*, seperti menambahkan node, menghapus node, serta mencari data pada daftar. Operasi-operasi tersebut mencakup penghapusan elemen pada posisi tertentu, di bagian awal, maupun akhir daftar. Seluruh proses ini diilustrasikan menggunakan bahasa pemrograman Java untuk memperlihatkan secara langsung bagaimana masing-masing operasi bekerja dalam struktur SLL.

Proses implementasinya dimulai dengan membentuk struktur *Singly Linked List* yang terdiri dari kumpulan objek node. Setelah struktur dasar terbentuk, dilakukan pengujian terhadap berbagai operasi seperti penghapusan node (baik di awal, akhir, maupun posisi tertentu) dan penambahan node di berbagai posisi. Tak hanya itu, mekanisme pencarian elemen dalam list juga dibahas agar dapat memberikan gambaran yang jelas tentang bagaimana data dalam *Singly Linked List* dapat dikelola secara fleksibel dan efisien.

B. Tujuan

Tujuan dari praktikum ini adalah:

1. Menjelajahi bagaimana elemen-elemen dalam Singly Linked List saling terhubung satu sama lain, di mana setiap node menyimpan data serta menunjuk ke node berikutnya dalam urutan.

2. Mempelajari cara membangun dan mengoperasikan struktur Singly Linked List, termasuk bagaimana menambahkan elemen, menghapus node, serta melakukan pencarian terhadap data yang tersimpan dalam daftar.

3. Memahami bagaimana struktur ini mendukung pengelolaan data secara dinamis dan efisien, di mana setiap perubahan pada elemen tertentu dapat dilakukan tanpa perlu memodifikasi keseluruhan susunan data.

C. Langkah-langkah pengerjaan

A. NodeSLL

1) Buatlah sebuah kelas baru bernama NodeSLL yang berfungsi sebagai representasi simpul dalam struktur Singly Linked List. Kelas ini harus memiliki dua atribut: satu untuk menyimpan nilai data pada simpul, dan satu lagi bernama next yang bertindak sebagai referensi menuju simpul berikutnya dalam daftar. Pada saat objek simpul dibuat, nilai next diatur ke null karena belum terhubung dengan simpul lain. Gunakan konstruktor NodeSLL (int data) untuk menginisialisasi objek simpul dengan nilai data yang diberikan, sekaligus menetapkan bahwa next bernilai null secara default.

```
1 package Pekan5;
2
3 public class NodeSLL {
4     int data;
5     NodeSLL next;
6     public NodeSLL(int data)
7     {
8         this.data = data;
9         this.next = null;
0     }
1 }
```

B. TambahSLL

1) Buatlah sebuah kelas baru dengan nama TambahSLL. Kelas ini akan berisi beberapa metode penting untuk mengelola Singly Linked List, antara lain: insertAtFront, insertAtEnd, GetNode, insertPos, printList, serta metode utama main.

2) Langkah pertama adalah membuat metode insertAtFront, yang digunakan untuk menambahkan simpul baru di bagian awal dari linked list. Caranya yaitu dengan membuat objek simpul baru (node) yang menyimpan nilai tertentu, kemudian mengatur referensi next dari node baru ini agar menunjuk ke simpul pertama (head) yang sudah ada.

```
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;
    }
```

3) Selanjutnya, implementasikan metode kedua yaitu insertAtEnd, yang berfungsi untuk menambahkan simpul baru di bagian akhir linked list. Buat sebuah simpul baru dengan variabel newNode dan tetapkan nilai yang akan disimpan. Jika daftar masih kosong (head bernilai null), maka simpul baru tersebut menjadi satu-satunya elemen. Namun jika tidak, lakukan iterasi dari simpul pertama hingga menemukan simpul terakhir (yaitu simpul dengan next == null), lalu tempatkan simpul baru setelahnya.

```
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;
}
```

4) Buat juga metode GetNode, yang bertugas untuk mengembalikan objek simpul baru berdasarkan nilai data yang diberikan. Metode ini berfungsi sebagai metode pembantu untuk menyederhanakan proses pembuatan simpul.

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

5) Implementasikan metode insertPos untuk menambahkan simpul pada posisi tertentu dalam linked list. Langkah-langkahnya adalah sebagai berikut: jika posisi yang diberikan kurang dari 1, tampilkan pesan "Invalid position". Jika posisi yang diminta adalah 1, maka node baru ditambahkan di awal daftar, seperti pada insertAtFront. Jika posisi lebih dari 1, lakukan perulangan untuk menelusuri daftar hingga posisi ke-(posisi - 1), lalu sisipkan node baru di posisi yang sesuai. Jika posisi tidak ditemukan dalam daftar, tampilkan pesan "Posisi di luar jangkauan".

```
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;
        }
  }
```

6) Buat metode printList yang digunakan untuk menampilkan seluruh isi linked list. Metode ini dilakukan dengan menelusuri daftar dari simpul pertama (head) dan mencetak setiap nilai node yang ditemukan, diikuti dengan tanda panah "-->". Jika telah mencapai simpul terakhir (next == null), cetak nilai terakhir dan akhiri dengan baris baru.

```
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();
  }
}
```

7) Pada metode main, buatlah sebuah linked list awal yang berisi nilai 2, 3, 5, dan 6, lalu tampilkan daftar tersebut. Selanjutnya, tambahkan simpul dengan nilai 1 di awal menggunakan insertAtFront, kemudian tampilkan daftar yang baru. Tambahkan juga simpul dengan nilai 7 di akhir daftar menggunakan insertAtEnd, lalu tampilkan hasilnya. Terakhir, tambahkan simpul dengan nilai 4 pada posisi ke-4 menggunakan insertPos, dan tampilkan hasil akhir dari daftar setelah seluruh penambahan selesai dilakukan.

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

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

    System.out.print("tambah 1 simpul di depan: ");
    int data = 1;
    head = insertAtFront(head, data);

    printList(head);
  System.out.print("tambah 1 simpul di belakang: ");
    int data2 = 7;
    head = insertAtEnd(head, data2);
    printList(head);
  System.out.print("tambah 1 simpul ke data 4: ");
    int data3 = 4;
    int pos=4;
    head = insertPos(head,pos,data3);
    printList(head);
}
```

8) Setelah seluruh metode selesai dibuat, jalankan program. Jika implementasi berhasil, maka output akan tampil sesuai dengan urutan perubahan yang dilakukan pada linked list.

```
<terminated> TambahSLL [Java Application] C:\Users\Lenovo\.p2\pool\pl
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 data 4: 1-->2-->3-->4-->5-->6-->7
```

C. PencarianSLL

1) Buatlah sebuah kelas baru bernama PencarianSLL, yang dirancang untuk melakukan proses pencarian dan penelusuran elemen pada struktur data Singly Linked List.

2) Di dalam kelas tersebut, tambahkan sebuah metode bernama searchKey. Metode ini bertugas mencari apakah terdapat simpul dalam linked list yang memiliki nilai data sama dengan nilai kunci (key) yang dicari. Proses pencarian dimulai dari simpul pertama (head), kemudian dilakukan pemeriksaan secara berurutan terhadap setiap simpul. Jika ditemukan simpul yang data-nya cocok dengan nilai key, maka metode akan mengembalikan nilai true, menandakan bahwa data ditemukan. Namun, jika pencarian mencapai akhir daftar (curr == null) tanpa menemukan kecocokan, maka metode akan mengembalikan nilai false.

```
package Pekan5;

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;   }
```

3) Tambahkan pula metode traversal, yang berfungsi untuk mencetak seluruh elemen yang terdapat dalam linked list. Proses dilakukan dengan memulai dari simpul pertama, lalu mencetak nilai data dari setiap simpul satu per satu. Iterasi berlanjut ke simpul berikutnya dengan mengikuti referensi next, hingga mencapai akhir daftar. Setelah semua data ditampilkan, tambahkan baris baru agar output lebih rapi dan mudah dibaca.

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

4) Selanjutnya, buat metode utama main. Pada metode ini, buat sebuah linked list yang terdiri dari lima simpul dengan nilai: 14, 21, 13, 30, dan 10. Setelah daftar dibuat, panggil metode traversal() untuk menampilkan seluruh isi daftar. Kemudian, tentukan nilai key yang akan dicari, yaitu 30, dan panggil metode searchKey() untuk memeriksa apakah nilai tersebut ada dalam linked list. Jika ditemukan, tampilkan pesan "Ketemu", dan jika tidak ditemukan, tampilkan pesan "Tidak ada".

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

5) Jalankan program yang telah dibuat. Jika implementasi berhasil, maka program pertama-tama akan menampilkan isi dari linked list. Setelah itu, proses pencarian akan dijalankan untuk mencari nilai 30. Karena nilai tersebut memang terdapat dalam daftar, maka output yang akan ditampilkan adalah "Ketemu".

```
<terminated> PencarianSLL [Java
Penelusuran SLL :  14
 21
 13
 30
 10

cari data 30 = ketemu
```

D. HapusSLL

1) Buatlah sebuah kelas bernama HapusSLL. Kelas ini akan memuat sejumlah metode yang berfungsi untuk menghapus simpul (node) dari struktur data Singly Linked List (SLL).

2) Tambahkan metode deleteHead, yang bertugas menghapus simpul pertama (head) dalam linked list. Logika metode ini adalah sebagai berikut: apabila linked list kosong (yaitu head bernilai null), maka kembalikan nilai null. Namun jika tidak kosong, ubah referensi head menjadi menunjuk ke simpul berikutnya (head = head.next). Hal ini akan secara efektif menghapus simpul pertama dari daftar. Setelah itu, kembalikan simpul head yang baru.

```
package Pekan5;

public class 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;
    }
```

3) Selanjutnya, buat metode removeLastNode untuk menghapus simpul terakhir pada linked list. Langkah-langkah yang dilakukan adalah: jika linked list kosong, kembalikan null; jika hanya ada satu simpul di dalam list, maka hapus simpul tersebut dan kembalikan null; namun jika terdapat lebih dari satu simpul, telusuri daftar hingga mencapai simpul kedua terakhir, lalu ubah referensinya (secondLast.next) menjadi null untuk memutus simpul terakhir. Terakhir, kembalikan head setelah penghapusan.

```
// fungsi menghapus node terakhir SLL
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;
}
```

4) Tambahkan pula metode deleteNode, yang digunakan untuk menghapus simpul pada posisi tertentu dalam linked list. Berikut logika kerjanya: jika linked list kosong, maka langsung kembalikan null. Jika posisi yang ingin dihapus adalah 0, maka simpul pertama dihapus dengan cara menggeser head ke simpul berikutnya. Jika posisi lebih dari 0, lakukan iterasi untuk mencapai simpul di posisi tersebut. Jika simpul ditemukan, ubah referensi simpul sebelumnya (prev.next) agar melewati simpul yang ingin dihapus. Jika posisi yang diminta tidak valid (tidak ada simpul di sana), tampilkan pesan "Data tidak ada".

```
// 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 null;

       // kasus 1: head dihapus
       if (position == 0) {
           head = head.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;
   }
```

5) Buat metode printList, yang berfungsi menampilkan seluruh isi linked list. Mulailah iterasi dari simpul head, dan cetak setiap nilai data yang ada, diikuti dengan simbol -->. Ketika simpul terakhir telah dicapai (yaitu referensi next bernilai null), cetak tulisan null sebagai penanda akhir dari daftar, lalu tambahkan baris baru.

```
// fungsi mencetak SLL
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();
}
```

6) Terakhir, buat metode utama main. Dalam metode ini, buatlah sebuah linked list awal dengan data sebagai berikut: 1 → 2 → 3 → 4 → 5 → 6 → null. Setelah itu, tampilkan isi awal dari linked list. Lanjutkan dengan menghapus simpul pertama menggunakan metode deleteHead(), lalu tampilkan hasil linked list setelah penghapusan. Selanjutnya, hapus simpul terakhir menggunakan metode removeLastNode(), dan tampilkan hasilnya. Terakhir, hapus simpul yang berada pada posisi ke-2 (dengan indeks mulai dari 0) menggunakan metode deleteNode(), lalu tampilkan kembali isi linked list setelah penghapusan.

```
// kelas main
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);
}
```

7) Setelah seluruh langkah di atas diimplementasikan, jalankan program. Jika program berhasil dijalankan dengan benar, maka hasil output akan menampilkan isi linked list setelah masing-masing proses penghapusan, sesuai dengan urutan yang telah dijelaskan.

```
<terminated> HapusSLL [Java Application] C:\Users\L
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

Singly Linked List (SLL) adalah struktur data yang efisien untuk mengelola data secara dinamis. Setiap node saling terhubung ke node berikutnya, sehingga penambahan atau penghapusan elemen bisa dilakukan tanpa perlu menggeser data seperti pada array.

Dari implementasi program, terlihat bahwa operasi dasar seperti menambah, menghapus, dan mencari node dapat dilakukan dengan mudah. Selain itu, SLL juga hemat memori karena hanya menyimpan referensi ke node berikutnya.

Secara keseluruhan, SLL sangat berguna untuk aplikasi yang membutuhkan manipulasi data yang fleksibel dan efisien.