LAPORAN PRAKTIKUM 5

STRUKTUR DATA

Oleh:
Ibrahim Mousa Dhani
2411532010

Dosen Pengampu : Dr. Wahyudi S.T, M.T.

Mata Kuliah : Struktur Data

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG

A. Pendahuluan

Dalam dunia struktur data, Linked List merupakan salah satu struktur linear yang digunakan untuk menyimpan data secara dinamis. Salah satu jenisnya adalah Singly Linked List (SLL), yaitu struktur data yang terdiri dari node-node di mana setiap node memiliki dua bagian: data dan pointer (penunjuk) ke node berikutnya. Sifat "singly" mengindikasikan bahwa setiap node hanya mengetahui alamat node selanjutnya, sehingga pergerakan atau traversal hanya dapat dilakukan dalam satu arah — dari kepala (head) ke ekor (tail)

Berbeda dengan array yang bersifat statis, Singly Linked List memungkinkan alokasi dan dealokasi elemen secara dinamis selama waktu eksekusi program. Oleh karena itu, SLL sangat berguna ketika ukuran data tidak diketahui di awal atau ketika operasi penyisipan dan penghapusan elemen perlu dilakukan dengan efisien.

Dalam praktikum ini, dilakukan implementasi kode program Java untuk mengelola sebuah Singly Linked List dengan beberapa operasi utama: penambahan node (di depan, belakang, dan posisi tertentu), penghapusan node (dari depan, belakang, dan posisi tertentu), pencarian nilai tertentu, serta penelusuran seluruh node. Setiap operasi diimplementasikan menggunakan fungsi-fungsi terpisah agar kode menjadi modular, mudah dipahami, dan dapat dikembangkan lebih lanjut

B. Tujuan Praktikum

1. Mengetahui dan memahami konsep dasar struktur data Singly Linked List (SLL) dan komponennya.
2. Mengimplementasikan operasi dasar SLL dalam bahasa pemrograman Java.
3. Memahami cara penambahan, dan penghapusan, node dalam linked list pada berbagai posisi (depan, belakang, dan posisi tertentu), serta penelusuran menggunakan traversal dan mencetak isi linked list
4. Meningkatkan pemahaman konsep pointer dan hubungan antar node dalam linked list.

C. Langkah Langkah

a. NodeSLL

1. Pertama buat package baru dan beri nama pekan5. Buat kelas baru dan beri nama NodeSLL. Kelas ini adalah struktur dasar yang nantinya akan digunakan oleh struktur Single Linked List yang lengkap

```
package pekan5;

public class NodeSLL {
```

2. Deklarasikan atribut ‘data’ bertipe int, atribut ini akan menyimpan nilai atau data yang dimiliki oleh node
3. Setelah itu deklarasikan juga atribut ‘next’. Tipe datanya adalah NodeSLL, artinya menunjuk ke objek lain dari kelas NodeSLL. Atribut ini adalah pointer ke node berikutnya.

```
// node bagian data
int data;
//Pointer ke node berikutnya
NodeSLL next;
```

4. Lalu buat Konstruktor, konstruktor merupakan method khusus yang dipanggil saat objek dibuat. Konstruktor ini menerima satu parameter yaitu ‘int data’

```
// Konstruktor menginisialisasi node dengan data
public NodeSLL (int data)
{
    this.data = data;
    this.next = null;
}
```

this.data = data; berfungsi untuk menyimpan nilai ke dalam atribut data milik objek.

this.next = null; artinya node ini belum menunjuk ke node lain, karena saat dibuat biasanya node berdiri sendiri dulu.

5. Berikut kode lengkapnya

```
1  package pekan5;
2
3  public class NodeSLL {
4      // node bagian data
5      int data;
6      //Pointer ke node berikutnya
7      NodeSLL next;
8      // Konstruktor menginisialisasi node dengan data
9      public NodeSLL (int data)
10     {
11         this.data = data;
12         this.next = null;
13     }
```

b. TambahSLL

1. Buat kelas baru dan beri nama TambahSLL. Gunakan NodeSLL yang sudah kamu buat sebelumnya sebagai struktur node-nya.

```
package pekan5;

public class TambahSLL {
```

2. Buat fungsi insertAtFront, untuk menambahkan node baru di awal (paling depan) dari linked list. Buat node baru dengan data value. Arahkan .next dari node baru ke head lama(NodeSLL). Return node baru sebagai head yang baru.

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

3. Lalu buat juga fungsi insertAtEnd, untuk menambahkan node baru di bagian akhir list. Buat node baru (newNode) dengan data value. Jika head == null, berarti list kosong dan kembalikan newNode sebagai node pertama. Kalau tidak kosong, telusuri node terakhir (yang next-nya null). Sambungkan node terakhir ke newNode.

```
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;
```

4. Selanjutnya buat fungsi GetNode yang bertujuan untuk membuat dan mengembalikan node baru.

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

5. Buat fungsi insertPos yang bertujuan untuk menambahkan node di posisi tertentu (bukan hanya di depan atau belakang). Jika posisi < 1 maka akan menampilkan error. Jika posisi = 1 akan menambahkan di depan. Jika posisi > 1 maka akan menelusuri list hingga ke posisi yang dimaksud. Terakhir sisipkan node di posisi tersebut.

```
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;
```

6. Buat fungsi printList untuk menampilkan isi linked list. Telusuri node satu per satu mulai dari head. Selama masih ada node berikutnya (curr.next != null), cetak data + panah -->. Jika sudah sampai node terakhir, cetak data tanpa panah.

```
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. Selanjutnya yang terakhir buat fungsi main(). Buat list awal terlebih dahulu yaitu 2→3→5→6 dan cetak/tampilkan ke layer

```
public static void main(String[] args) {
    // buat linked list 2 -> 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
    System.out.print("Senarai berantai awal: ");
    printList(head);
```

8. Lalu tambahkan node baru di bagian depan(head) dan tampilkan ke layar

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

9. Tambahkan node baru di bagian belakang lalu tampilkan

```
System.out.print("Tambah 1 simpul di belakang: ");
int data2 = 7;
head = insertAtEnd(head, data2);
// cetak update list
printList(head);
```

10. Terakhir tambahkan node baru di posisi ke-4 dan tampilkan hasilnya

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

11. Setelah semua benar dan tidak ada eror, maka jalankan program, lalu akan menghasilkan output sebagai 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 data 4: 1--> 2--> 3--> 4--> 5--> 6--> 7
```

12. Berikut kode program versi lengkapnya

```
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 di akhir
    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;
    }

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

```
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();
        }
    }
public static void main(String[] args) {
    // buat linked list 2 -> 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
    System.out.print("Senarai berantai awal: ");
    printList(head);

    // tambahkan node baru di depan
    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
    System.out.print("Tambah 1 simpul di belakang: ");
    int data2 = 7;
    head = insertAtEnd(head, data2);
    // cetak update list
    printList(head);

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

c. HapusSLL

1. Buat kelas baru dan beri nama HapusSLL. Gunakan NodeSLL yang sudah kamu buat sebelumnya sebagai struktur node-nya.

```
package pekan5;

public class HapusSLL {
```

2. Buat fungsi deletedHead yang berfungsi untuk menghapus node paling depan dari linked list. Cek apakah linked list kosong (head == null). Jika iya, kembalikan null. Geser pointer head ke node berikutnya (head = head.next). dan kembalikan head baru

```
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. Buat fungsi removeLastNode untuk menghapus node terakhir dari linked list. Jika list kosong → return null. Jika hanya ada 1 node → return null. Jika lebih dari 1 node: Telusuri hingga node ke-dua dari belakang. Set .next dari node itu ke null, agar node terakhir dihapus. Return head yang tidak berubah.

```
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. Buat fungsi deleteNode(NodeSLL head, int position) untuk menghapus node pada posisi tertentu. Cek jika linked list kosong maka return head tanpa perubahan. Jika position == 1 maka hapus head (geser head ke head.next). Jika bukan di awal: Telusuri node hingga posisi tersebut. Simpan node sebelumnya

(prev). Jika ditemukan node di posisi itu, hubungkan prev.next ke temp.next. Jika tidak ditemukan maka cetak pesan “Data tidak ada”.

```
public static NodeSLL deleteNode(NodeSLL head, int position) {
    NodeSLL temp = head;
    NodeSLL prev = null;
    // jika linked list null
    if (temp == null)
        return head;
    // kasus 1: head dihapus
    if (position == 1) {
        head = temp.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 fungsi printList(NodeSLL head) untuk menampilkan isi linked list ke layar. Mulai dari head, telusuri semua node. Cetak data tiap node dengan tanda panah --> antar node. Cetak node terakhir tanpa panah.

```
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();
```

6. Selanjutnya buat fungsi main(), dan buat list awal SLL terlebih dahulu yaitu 1→2→3→4→5→6 dan cetak/tampilkan ke layer

```
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);
```

7. Lalu hapus node di bagian depan(head) dan tampilkan ke layer

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

8. Hapus node bagian akhir/belakang lalu tampilkan

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

9. Terakhir hapus node di posisi ke-2 (yaitu 3) dan tampilkan hasilnya

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

10. Setelah semua benar dan tidak ada eror, maka jalankan program, lalu akan menghasilkan output sebagai berikut

```
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. Berikut kode program versi lengkapnya

```
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; }
    // 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; }

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

    // fungsi mencetak SLL
    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();
    }

//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);
         // hapus simpul pada posisi ke-2
         int position = 2;
         head = deleteNode(head, position);
         System.out.println("List setelah posisi 2 dihapus: ");
         printList(head);
```

d. PencarianSLL

1. Buat kelas baru pada pa dan beri nama PencarianSLL. Gunakan NodeSLL yang sudah kamu buat sebelumnya sebagai struktur node-nya

```
package pekan5;
    public class PencarianSLL {
```

2. Selanjutnya buat fungsi searchKey(NodeSLL head, int key) untuk mencari apakah suatu data ada di dalam linked list. Terima parameter: NodeSLL head: node pertama dari linked list. int key: data yang dicari. Gunakan while loop untuk menelusuri semua node: Jika data (curr.data) sama dengan key, return true. Jika tidak, lanjut ke node berikutnya (curr = curr.next). Jika seluruh list telah diperiksa dan key tidak ditemukan, return false.

```
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. Buat fungsi traversal (NodeSLL head) untuk menampilkan isi lengkap linked list. Mulai dari head. Selama curr tidak null: Cetak nilai curr.data ke layar. Pindah ke node berikutnya. Setelah selesai, pindah baris (dengan System.out.println()).

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

4. Setelah itu buat buat fungsi main(), lalu buat linked list awal terlebih dahulu yaitu seperti berikut 14→21→13→30→20

```
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);
```

5. Lalu tampilkan isi linked list

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

6. Selanjutnya tentukan data yang ingin dicari dengan int key, lalu panggil fungsi searchKey() yang sudah dibuat tadi, jika key ditemukan maka tampilkan "ketemu". Dan jika tidak maka tampilkan "tidak ada".

```
int key = 30;
System.out.print("cari data " + key + " = ");
if (searchKey(head, key))
    System.out.println("ketemu");
else
    System.out.println("tidak ada");}}
```

7. Setelah semua benar dan tidak ada eror, maka jalankan program, lalu akan menghasilkan output sebagai berikut

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

8. Berikut kode program versi lengkapnya

```
1  package pekan5;
2      public class PencarianSLL {
3
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;  }
11         public static void traversal(NodeSLL head) {
12             // mulai dari head
13             NodeSLL curr = head;
14             // telusuri sampai pointer null
15             while (curr != null) {
16                 System.out.print(" " + curr.data);
17                 curr = curr.next;}
18             System.out.println();
19             }
20         public static void main(String[] args) {
21             NodeSLL head = new NodeSLL(14);
22             head.next = new NodeSLL(21);
23             head.next.next = new NodeSLL(13);
24             head.next.next.next = new NodeSLL(30);
25             head.next.next.next.next = new NodeSLL(10);
26             System.out.print("Penelusuran SLL : ");
27             traversal(head);
28             // data yang akan dicari
29             int key = 30;
30             System.out.print("cari data " + key + " = ");
31             if (searchKey(head, key))
32                 System.out.println("ketemu");
33             else
34                 System.out.println("tidak ada");}}
```

D. Kesimpulan

Berdasarkan praktikum yang telah dilakukan kita dapat mengetahui dan memahami bahwa Singly Linked List (SSL) merupakan struktur data dinamis yang efisien untuk penambahan dan penghapusan elemen tanpa perlu menggeser elemen lain, seperti pada array.

Operasi dasar seperti penyisipan (insertion), penghapusan (delete), penelusuran (traversal), dan pencarian (searching) dapat diimplementasikan dengan logika pemrograman yang cukup sederhana namun membutuhkan pemahaman yang baik terhadap penggunaan pointer (next).

Pemahaman terhadap hubungan antar node sangat penting agar tidak terjadi kesalahan seperti kehilangan referensi node. Dengan latihan implementasi ini, mahasiswa dapat memahami cara kerja linked list dan pentingnya struktur data ini dalam pengembangan perangkat lunak yang melibatkan pengelolaan data secara dinamis.