LAPORAN PRAKTIKUM PEKAN 6 STRUKTUR DATA
DOUBLE LINKED LIST (DLL)

OLEH :

FADIL INSANUS SIDDIK

(2411532013)

MATA KULIAH : PRAKTIKUM STRUKTURR DATA

DOSEN PENGAMPU : Dr. Wahyudi, S.T, M.T.

FAKULTAS TEKNOLOGI INFORMASI

PROGRAM STUDI INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, MEI 2025

A. PENDAHULUAN

Struktur data merupakan salah satu komponen penting dalam pemrograman yang berfungsi untuk menyimpan dan mengatur data secara efisien. Salah satu bentuk struktur data yang sering digunakan adalah *linked list*. Pada praktikum ini, fokus utama adalah pada **Double Linked List (DLL)** atau **linked list ganda**, yaitu struktur data linier yang terdiri dari rangkaian node di mana setiap node memiliki dua buah referensi: satu menunjuk ke node berikutnya (*next*) dan satu lagi menunjuk ke node sebelumnya (*prev*).

Keunggulan dari Doubly Linked List dibandingkan dengan Single Linked List adalah kemampuannya untuk ditelusuri dari dua arah, baik dari awal ke akhir (*forward traversal*) maupun dari akhir ke awal (*backward traversal*). Hal ini memungkinkan operasi seperti penambahan, penghapusan, dan pencarian data menjadi lebih fleksibel. Selain itu, DLL sangat berguna ketika dibutuhkan navigasi dua arah, misalnya dalam implementasi struktur data seperti *deque*, *editor undo/redo*, dan *navigasi riwayat*.

Dalam praktikum ini, mahasiswa mempelajari bagaimana membuat struktur DLL secara manual di dalam bahasa pemrograman Java, serta mengimplementasikan berbagai operasi penting seperti penghapusan node di awal, akhir, atau posisi tertentu, dan melakukan penelusuran data baik secara maju maupun mundur.

B. TUJUAN PRAKTIKUM

1. Memahami konsep dasar dan karakteristik dari struktur data Double Linked List (DLL).
2. Mampu mengimplementasikan struktur node yang memiliki referensi ganda (*next* dan *prev*).
3. Mampu mengimplementasikan operasi dasar pada DLL, seperti:
   - Penghapusan node di awal (delHead)
   - Penghapusan node di akhir (delLast)
   - Penghapusan node pada posisi tertentu (delPos)
   - Penelusuran maju (forwardTraversal)
   - Penelusuran mundur (backwardTraversal)
4. Melatih kemampuan berpikir logis dan pemahaman alur data dalam struktur non-sekuensial.
5. Mengembangkan keterampilan pemrograman Java dalam konteks manipulasi struktur data.

C. LANGKAH-LANGKAH

1. NodeDLL

```
1 package Pekan6;
```

- Fungsi: Menyatakan bahwa file/class ini berada dalam package bernama Pekan6.
- Tujuan: Mengelompokkan kelas-kelas dalam satu folder/module yang sama.

```
public class NodeDLL {
```

- Fungsi: Mendefinisikan sebuah kelas publik bernama NodeDLL.
- Tujuan: Kelas ini mewakili satu node pada struktur Double Linked List (DLL).

```
5       int data; //data
```

- Fungsi: Mendeklarasikan variabel data bertipe int.
- Tujuan: Menyimpan nilai utama dalam node.

```
NodeDLL next; //Pointer ke next node
```

- Fungsi: Mendeklarasikan referensi next ke node berikutnya.
- Tujuan: Menghubungkan node ini ke node setelahnya (forward link).

```
7       NodeDLL prev; //Pointer ke previous node
```

- Fungsi: Mendeklarasikan referensi prev ke node sebelumnya.
- Tujuan: Menghubungkan node ini ke node sebelumnya (backward link).

```
public NodeDLL(int data) {
```

- Fungsi: Konstruktor kelas NodeDLL, akan dipanggil saat membuat objek baru.
- Parameter: data – nilai integer yang ingin disimpan di node.

```
this.data = data;
```

- Fungsi: Menetapkan nilai yang dikirim ke konstruktor ke properti data milik objek ini.

```
this.next = null;
```

- Fungsi: Menginisialisasi pointer next dengan null, artinya belum terhubung ke node mana pun.

```
this.prev = null;
```

- Fungsi: Menginisialisasi pointer prev dengan null, artinya belum ada node sebelumnya.

2. InsertDLL

```
3 public class InsertDLL {
```

- Mendefinisikan class dengan nama InsertDLL

```
5⊖    static NodeDLL insertBegin(NodeDLL head, int data) {
```

- Mendefinisikan method statis `insertBegin`, menerima parameter `head` (simpul awal) dan `data`.

```
    NodeDLL new_node = new NodeDLL(data);
```

- Membuat simpul baru berisi data.

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

- Menghubungkan next dari simpul baru ke head.

```
if (head != null) {
    head.prev = new_node;
}
```

- Jika hea dtidak kosong, atur prev dari simpul awal ke simpul baru.

```
return new_node;
```

- Kembalikan simpul baru sebagai head baru.

```
public static NodeDLL insertEnd(NodeDLL head, int newData) {
```

- Metode untuk menambahkan simpul di akhir DLL.

```
NodeDLL newNode = new NodeDLL(newData);
```

- Buat simpul baru.

```
if (head == null) {
    head = newNode;
}
```

- Jika DLL kosong, simpul baru jadi head.

```
else {
    NodeDLL curr = head;
    while (curr.next!= null) {
        curr = curr.next;
    }
    curr.next = newNode;
    newNode.prev = curr;
}
```

- Kalau tidak kosong, telusuri ke simpul terakhir, sambungkan simpul baru ke akhir.

```
return head;
```

- Kembalikan kepala DLL.

```
public static NodeDLL insertAtPosition(NodeDLL head, int pos, int new_data) {
```

- Menyisipkan simpul di posisi tertentu.

```
NodeDLL new_node = new NodeDLL(new_data);
```

- Buat simpul baru.

```
if (pos == 1) {
    new_node.next = head;
    if (head != null) {
        head.prev = new_node; }
    head = new_node;
    return head;}
```

- Jika posisi 1, lakukan seperti insertBegin.

```
    NodeDLL curr = head;
for (int i = 1; 1 < pos - 1 && curr != null; ++i) {
    curr = curr.next;}
```

- Telusuri ke simpul sebelum posisi yang dituju.

```
if (curr == null) {
    System.out.println("Posisi tidak ada");
    return head;}
```

- Jika simpul target tidak ada, batalkan.

```
new_node.prev = curr;
new_node.next = curr.next;
curr.next = new_node;
if (new_node.next != null) {
    new_node.next.prev = new_node; }
```

- Masukkan simpul di tengah, sambungkan next dan prev.

```
return head;
}
```

- Kembalikan head.

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

- Menampilkan DLL

```
    NodeDLL curr = head;
    while (curr != null) {
        System.out.println(curr.data + " <-> ");
        curr = curr.next;
    }
    System.out.println();
```

- Cetak isi simpul dari awal sampai akhir.

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

- Fungsi utama pada class ini, program dimulai.

```
  NodeDLL head = new NodeDLL(2);
  head.next = new NodeDLL(3);
  head.next.next = new NodeDLL(5);
  head.next.next.prev = head.next;
```

- Membuat DLL awal: 2 <-> 3 <-> 5

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

- Cetak kondisi awal DLL.

```
  head = insertBegin(head, 1);
  System.out.print("simpul 1 ditambah di awal: ");
  printList(head);
```

- Tambah simpul 1 di awal.

```
System.out.print("simpul 6 ditambah di akhir: ");
int data = 6;
head = insertEnd(head, data);
printList(head);
```

- Tambah simpul 6 di akhir.

```
System.out.print("tambah node 4 di posisi 4: ");
int data2 = 4;
int pos = 4;
head = insertAtPosition(head, pos, data2);
printList(head);
```

- Tambah simpul 4 di posisi ke 4.

3. HapusDLL

```
3 public class HapusDLL {
```

- Mendeklarasikan class dengan nama HapusDLL

```
 public static NodeDLL delHead(NodeDLL head) {
```

- Fungsi menghapus node pertama

```
  if (head== null) {
      return null; }
```

- Jika list kosong, langsung return null.

```
NodeDLL temp = head;
head = head.next;
```

- Simpan node awal, lalu pindahkan head ke node berikutnya.

```
if (head != null) {
    head.prev = null; }
```

- Jika node berikutnya ada, hapus link ke node sebelumnya (hapus referensi ke node yang akan dihapus).

```
return head;
}
```

- Kembalikan head baru.

```
public static NodeDLL delLast(NodeDLL head) {
```

- Fungsi menghapus node terakhir.

```
if (head == null) {
    return null; }
if (head.next == null) {
    return null; }
```

- Jika list kosong atau hanya satu node, tidak bisa hapus node terakhir → return null.

```
NodeDLL curr = head;
while (curr.next != null) {
    curr = curr.next;
}
```

- Loop sampai node terakhir.

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

- Putus link ke node terakhir dari node sebelumnya.

```
return head;
```

- Kembalikan head list.

```
public static NodeDLL delPos(NodeDLL head, int pos) {
```

- Fungsi menghapus node pada posisi tertentu

```
if (head== null) {
    return head;}
```

- Jika kosong, kembalikan list tanpa perubahan.

```
NodeDLL curr = head;
for (int i = 1; curr != null && i < pos; ++i) {
    curr = curr.next; }
```

- Loop ke posisi yang ingin dihapus.

```
if (curr == null) {
    return head;}
```

- Kalau posisi tidak valid (lebih dari panjang list), return tanpa perubahan.

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

- Putus link dari node sebelumnya ke node setelah curr.

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

- Putus link dari node sesudah ke node sebelumnya curr.

```
if (head == curr) {
    head = curr.next;    }
```

- Jika node yang dihapus adalah head, pindahkan head ke node berikutnya.

```
return head;
```

- Return head baru.

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

- Fungsi mencetak DLL

```
NodeDLL curr = head;
while (curr != null) {
    System.out.print(curr.data + " ");
    curr = curr.next;
}
System.out.println();
}
```

- Loop sambil mencetak data dari setiap node sampai habis.

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

- Fungsi utama pada class ini, program di mulai.

```
NodeDLL head = new NodeDLL(1);
head.next = new NodeDLL(2);
head.next.prev = head;
head.next.next = new NodeDLL(3);
head.next.next.prev = head.next;
head.next.next.next = new NodeDLL(4);
head.next.next.next.prev = head.next.next;
head.next.next.next.next = new NodeDLL(5);
head.next.next.next.next.prev = head.next.next.next;
```

- Inisialisasi simpul dari hingga penambahan node ke-5.

```
System.out.print("DLL awal: ");
printList(head);
```

- Cetak list awal.

```
System.out.print("Setelah head dihapus: ");
head = delHead(head);
printList(head);
```

- Hapus node awal dan cetak.

```
System.out.print("Setelah node terakhir dihapus: ");
head = delLast(head);
printList(head);
```

- Hapus node akhir dan cetak.

```
System.out.print("menghapus node ke 2: ");
head = delPos(head, 2);
printList(head);
```

- Hapus node ke-2 dan cetak.

4. PenelusuranDLL

```
public class PenelusuranDLL {
```

- Mendeklarasikan class dengan nama PenelusuranDLL

```
static void forwardTraversal(NodeDLL head) {
    NodeDLL curr = head;
```

- Deklarasi method forwardTraversal, yang menerima node awal (head). Pointer curr digunakan untuk iteras

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

- Loop selama curr tidak null. Cetak data dan arah panah, kemudian lanjut ke node berikutnya (curr.next).

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

- Pindah baris setelah semua data dicetak.

```
static void backwardTraversal(NodeDLL tail) {
    NodeDLL curr = tail;
```

- Method untuk menelusuri DLL dari node akhir (tail) ke belakang. Pointer curr diset ke tail.

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

- Loop mundur selama curr tidak null. Cetak data dan panah, lalu ke node sebelumnya (curr.prev).

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

- Pindah baris setelah selesai mencetak.

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

- Method utama program Java. Eksekusi dimulai dari sini.

```
NodeDLL head = new NodeDLL(1);
NodeDLL second = new NodeDLL(2);
NodeDLL third = new NodeDLL(3);
```

- Membuat 3 node baru berisi data 1, 2, dan 3.

```
head.next = second;
second.prev = head;
```

- Menghubungkan head ke secoong (arah maju) dan second ke head (arah mundur).

```
second.next = third;
third.prev = second;
```

- Menghubungkan second ke third (maju) dan third ke second (mundur).

```
System.out.println("Penelusuran maju: ");
forwardTraversal(head);
```

- Mencetak header dan memanggil method forwardTraversal untuk mencetak semua data dari depan ke belakang.

```
System.out.println("Penelusuran mundur: ");
backwardTraversal(third);
```

- Mencetak third dan memanggil method backwardTraversal untuk mencetak semua data dari belakang ke depan.