LAPORAN PRATIKUM STRUKTUR DATA PEKAN 6
OPERASI DOUBLY LINKED LIST (DLL)

MATA KULIAH

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

OLEH:

AUFAN TAUFIQURRAHMAN

NIM 2411532011

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

2025

A. PENDAHULUAN

Doubly Linked List (DLL) adalah struktur data linear yang terdiri dari node-node terhubung dua arah (maju dan mundur). Setiap node menyimpan data, pointer ke node berikutnya (next), dan pointer ke node sebelumnya (prev). Praktikum ini bertujuan mengimplementasikan operasi dasar DLL: penyisipan, penghapusan, dan penelusuran dua arah.

B. TUJUAN

- Memahami struktur node DLL (prev, data, next).
- Mengimplementasikan operasi penyisipan node di depan, belakang, dan posisi tertentu.
- Mengimplementasikan operasi penghapusan node di head, tail, dan posisi tertentu.
- Menguji penelusuran dua arah (maju dan mundur).

C. PROSEDUR

1. Class NodeDLL.java

```java
package pekan6;

public class NodeDLL {

    // Aufan_Taufiqurrahman
    // 2411532011

    int data;
    NodeDLL next;
    NodeDLL prev;

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

- public class NodeDLL → Deklarasi class node untuk struktur Double Linked List.
- int data → Menyimpan nilai (data) dari node.

- NodeDLL prev, next → Mewakili pointer ke node sebelumnya dan node sesudahnya.
- public NodeDLL(int data) → Konstruktor untuk membuat node baru dengan nilai data tertentu.
- this.data = data → Menetapkan nilai data pada node sesuai parameter input.
- this.prev = null → Inisialisasi pointer prev dengan null karena node belum punya tetangga kiri.
- this.next = null → Inisialisasi pointer next dengan null karena node belum punya tetangga kanan.

2. Class InsertDLL.java
   a. Penyisipan di Depan (insertBegin)

```
static NodeDLL insertBegin (NodeDLL head, int data) {
    NodeDLL new_node = new NodeDLL(data);
    new_node.next = head;
    if (head != null) {
        head.prev = new_node;
    }
    return new_node;
}
```

   - NodeDLL newNode = new NodeDLL(data) → Membuat node baru dengan nilai data.
   - if (head == null) → Mengecek apakah list masih kosong.
   - head = newNode → Jika kosong, tetapkan newNode sebagai head.
   - else → Jika list tidak kosong:
   - newNode.next = head → Hubungkan node baru ke node head saat ini.
   - head.prev = newNode → Hubungkan pointer prev dari node head ke node baru.
   - head = newNode → Jadikan newNode sebagai node head yang baru.

   b. Penyisipan di Belakang (insertEnd)

```
public static NodeDLL insertEnd(NodeDLL head, int newData) {
    NodeDLL newNode = new NodeDLL (newData);
    if (head == null) {
        head = newNode;
    } else {
        NodeDLL curr = head;
        while (curr.next != null) {
            curr = curr.next;
        }
        curr.next = newNode;
        newNode.prev = curr;
    }
    return head;
}
```

- NodeDLL newNode = new NodeDLL(data) → Buat node baru dengan nilai data.
- if (head == null) → Jika list kosong:

- head = newNode → Jadikan newNode sebagai head.
- else → Jika list tidak kosong:
- NodeDLL curr = head → Pointer bantu untuk iterasi dari awal.
- while (curr.next != null) → Iterasi hingga mencapai node terakhir.
- curr = curr.next → Geser pointer ke node berikutnya.
- curr.next = newNode → Sambungkan node terakhir ke node baru
- newNode.prev = curr → Sambungkan node baru ke node terakhir melalui prev.

c. Penyisipan di Posisi Tertentu (insertAtPosition)

```
public static NodeDLL insertAtPosition (NodeDLL head, int pos, int new_data) {
    NodeDLL new_node = new NodeDLL(new_data);
    if (pos == 1) {
        new_node.next = head;
        if (head != null) {
            head.prev = new_node;
        }
        head = new_node;
        return head;
    }
    NodeDLL curr = head;
    for (int i = 1; i < pos - 1 && curr != null; ++i) {
        curr = curr.next;
    }
    if (curr == null) {
        System.out.println(x:"Posisi tidak ada");
        return head;
    }

    new_node.prev = curr;
    new_node.next = curr.next;
    curr.next = new_node;

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

- NodeDLL newNode = new NodeDLL(data) → Buat node baru dengan nilai data.
- if (pos == 1) → Jika posisi yang diminta adalah pertama:
- insertBegin(data) → Panggil method insert di awal.
- else → Jika posisi bukan pertama:
- NodeDLL curr = head → Pointer bantu dari head.
- for (int i = 1; i < pos - 1; i++) curr = curr.next → Iterasi ke posisi sebelum posisi tujuan.
- newNode.next = curr.next → Hubungkan next node baru ke node di posisi saat ini.
- if (curr.next != null) → Jika node berikutnya bukan null:
- curr.next.prev = newNode → Hubungkan prev dari node berikut ke node baru.
- curr.next = newNode → Hubungkan node saat ini ke node baru.
- newNode.prev = curr → Hubungkan node baru ke node sebelumnya (curr).

d. Mencetak nilai node (printList)

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

- NodeDLL curr = head → Mulai dari node pertama.
- while (curr != null) → Selama masih ada node:
- System.out.print(curr.data + " ") → Cetak nilai node.
- curr = curr.next → Geser ke node berikutnya.

e. Method main

```
Run | Debug | Run main | Debug main
public static void main(String[] args) {
    System.out.println(x:"Aufan Taufiqurrahman");
    System.out.println(x:"2411532011");

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

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

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

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

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

- InsertDLL list = new InsertDLL() → Membuat objek InsertDLL.
- list.insertEnd(10) → Tambah node 10 di akhir.
- list.insertEnd(20) → Tambah node 20 di akhir.
- list.insertEnd(30) → Tambah node 30 di akhir.
- list.insertBegin(5) → Tambah node 5 di awal.

- list.insertPosition(3, 15) → Tambah node 15 di posisi ke-3.
- list.printList() → Cetak seluruh isi list dari awal ke akhir.

3. Class HapusDLL,java

```
public static void main(String[] args) {
    System.out.println(x:"Aufan Taufiqurrahman");
    System.out.println(x:"2411532011");
    NodeDLL head = new NodeDLL(data:1);
    head.next = new NodeDLL(data:2);
    head.next.prev = head;
    head.next.next = new NodeDLL(data:3);
    head.next.next.prev = head.next;
    head.next.next.next = new NodeDLL(data:4);
    head.next.next.next.prev = head.next.next;
    head.next.next.next.next = new NodeDLL(data:5);
    head.next.next.next.next.prev = head.next.next.next;

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

    System.out.print(s:"Stelah head dihapus: ");
    head = delHead(head);
    printList(head);

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

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

    printList(head);
}
```

a. delHead()

```
public static NodeDLL delHead (NodeDLL head) {
    if (head == null) {
        return null;
    }
    NodeDLL temp = head;
    head = head.next;
    if (head != null) {
        head.prev = null;
    }
    return head;
}
```

- if (head == null)

  → Mengecek apakah linked list kosong.
- return

  → Jika kosong, keluar dari method tanpa melakukan apa-apa.
- if (head.next == null)

  → Mengecek apakah hanya ada satu node di dalam list.
- head = null

  → Menghapus satu-satunya node yang ada.
- else

  → Jika lebih dari satu node:
- head = head.next

  → Memindahkan pointer head ke node berikutnya.
- head.prev = null

  → Menghapus koneksi ke node lama (yang dihapus) dari arah belakang.

b. delLast()

```java
public static NodeDLL delLast (NodeDLL head) {
    if (head == null) {
        return null;
    }
    if (head.next == null) {
        return null;
    }
    NodeDLL curr = head;
    while (curr.next != null) {
        curr = curr.next;
    }

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

- if (head == null) → Mengecek apakah linked list kosong.
- return → Jika kosong, keluar dari method.
- if (head.next == null) → Mengecek apakah hanya ada satu node.
- head = null → Menghapus node tersebut.
- Else → Jika list memiliki lebih dari satu node:
- NodeDLL curr = head → Pointer bantu untuk iterasi dari awal.
- while (curr.next != null) → Iterasi hingga mencapai node terakhir.
- curr = curr.next → Pindah ke node berikutnya.
- curr.prev.next = null → Menghapus node terakhir dengan memutus koneksi dari node sebelumnya.

c. delPost(int pos)

```java
public static NodeDLL delPos(NodeDLL head, int pos) {
    if (head == null) {
        return head;
    }
    NodeDLL curr = head;

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

- if (head == null || pos <= 0)⟶ Mengecek apakah list kosong atau posisi tidak valid.
- Return ⟶ Keluar dari method jika kondisi tidak memenuhi.
- NodeDLL curr = head⟶ Pointer bantu untuk iterasi.
- for (int i = 1; curr != null && i < pos; i++)⟶ Iterasi ke posisi yang ditentukan.
- curr = curr.next ⟶ Pindahkan pointer ke node berikutnya di setiap iterasi.
- if (curr == null)⟶ Jika node tidak ditemukan (posisi terlalu besar), keluar dari method.
- if (curr.next != null) ⟶ Jika node yang dihapus bukan yang terakhir:
- curr.next.prev = curr.prev ⟶ Hubungkan node setelah curr ke node sebelum curr.
- if (curr.prev != null) ⟶ Jika node yang dihapus bukan yang pertama:
- curr.prev.next = curr.next ⟶ Hubungkan node sebelum curr ke node setelah curr.
- else⟶ Jika node yang dihapus adalah node pertama:
- head = curr.next ⟶ Pindahkan head ke node berikutnya.

d. printList()

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

- NodeDLL curr = head

  → Pointer bantu untuk iterasi dari awal.
- while (curr != null)

  → Iterasi selama node masih ada.
- System.out.print(curr.data + \" \")

  → Cetak nilai data dari node saat ini.
- curr = curr.next

  → Geser ke node berikutnya.

4. Class PenelusuranDLL.java

a. forwardTraversal

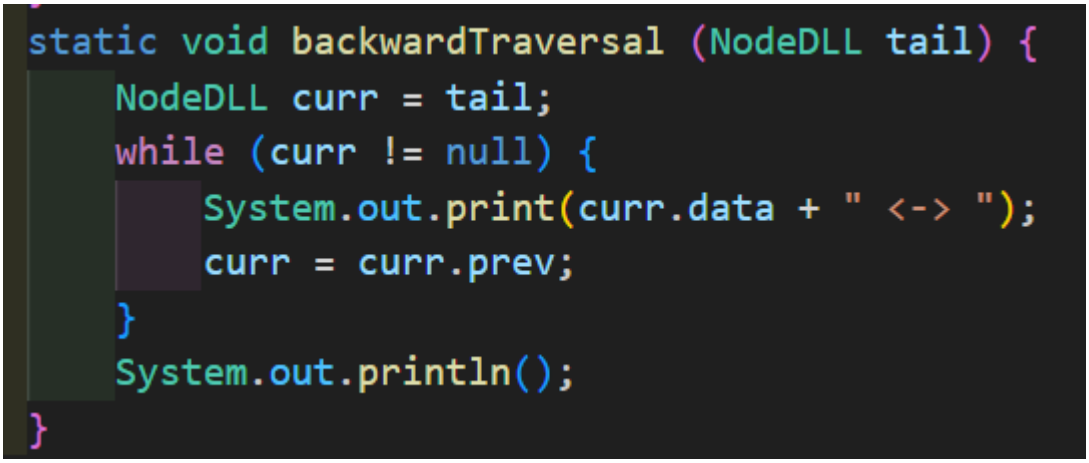
```
static void ForwardTraversal(NodeDLL head) {
    NodeDLL curr = head;

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

- NodeDLL curr = head

  → Menginisialisasi pointer curr agar mulai dari node head (awal list).

- System.out.println("Penelusuran dari awal ke akhir:")
  → Menampilkan teks penanda bahwa traversal akan dilakukan maju.
- while (curr != null)
  → Melakukan iterasi selama node saat ini tidak null (masih ada node berikutnya).
- System.out.print(curr.data + " ")
  → Menampilkan nilai data dari node yang sedang diakses.
- curr = curr.next
  → Melanjutkan pointer ke node berikutnya.

b. backwardTraversal



```
static void backwardTraversal (NodeDLL tail) {
    NodeDLL curr = tail;
    while (curr != null) {
        System.out.print(curr.data + " <-> ");
        curr = curr.prev;
    }
    System.out.println();
}
```

- NodeDLL curr = tail
  → Menginisialisasi pointer curr agar mulai dari node terakhir (tail).
- System.out.println("Penelusuran dari akhir ke awal:")
  → Menampilkan teks penanda bahwa traversal akan dilakukan mundur.
- while (curr != null)
  → Melakukan iterasi selama node saat ini tidak null (masih ada node sebelumnya).
- System.out.print(curr.data + " ")
  → Menampilkan nilai data dari node yang sedang diakses.
- curr = curr.prev
  → Melanjutkan pointer ke node sebelumnya.

c. Method Main

```
Run | Debug | Run main | Debug main
public static void main(String[] args) {
    System.out.println(x:"Aufan Taufiqurrahman");
    System.out.println(x:"2411532011");

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

    head.next = second;
    second.prev = head;
    second.next = third;
    third.prev = second;

    System.out.println(x:"Penelusuran maju:");
    ForwardTraversal(head);

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

- NodeDLL head = new NodeDLL(10)
  → Membuat node pertama (head) dengan nilai data 10.
- NodeDLL second = new NodeDLL(20)
  → Membuat node kedua dengan nilai data 20.
- NodeDLL third = new NodeDLL(30)
  → Membuat node ketiga dengan nilai data 30.
- head.next = second
  → Menyambungkan head ke node kedua melalui pointer next.
- second.prev = head
  → Menyambungkan node kedua ke head melalui pointer prev.
- second.next = third
  → Menyambungkan node kedua ke node ketiga melalui pointer next.
- third.prev = second
  → Menyambungkan node ketiga ke node kedua melalui pointer prev.
- forwardTraversal(head)
  → Memanggil method untuk menelusuri list dari depan ke belakang.
- backwardTraversal(third)
  → Memanggil method untuk menelusuri list dari belakang ke depan.