

LAPORAN PRAKTIKUM STRUKTUR DATA
DOUBLY LINKED LIST



OLEH :

JOVANTRI IMMANUEL GULO

NIM 2411532014

MATA KULIAH STRUKTUR DATA

DOSEN PENGAMPU :

Dr. Ir. Wahyudi, S.T, M.T

FAKULTAS TEKNOLOGI INFORMASI

DEPARTEMEN INFORMATIKA

UNIVERSITAS ANDALAS

PADANG, 27 MEI 2025

A. Pendahuluan

Doubly linked list merupakan pengembangan dari struktur linked list yang memungkinkan setiap node memiliki dua pointer—satu menunjuk ke node sebelumnya dan satu lagi ke node berikutnya. Dengan struktur ini, traversal data dapat dilakukan dalam kedua arah, baik maju maupun mundur, sehingga memberikan fleksibilitas lebih dibandingkan singly linked list. Kali ini, praktikum yang akan dilaksanakan ini bertujuan untuk memahami cara kerja doubly linked list serta mengimplementasikan berbagai operasinya, seperti penyisipan, penghapusan, dan pencarian data.

B. Tujuan

Tujuan dari dilakukannya praktikum ini adalah

1. Mampu memahami cara mengimplementasikan doubly linked list.
2. Mampu menggunakan konstruktor dan method pada doubly linked list.
3. Mampu mengimplementasikan doubly linked list dalam kehidupan sehari-hari.
4. Mampu memahami cara kerja node head, next, dan prev pada doubly linked list.

C. Langkah – langkah Pengerjaan

Berikut adalah langkah-langkah dalam pengerjaan praktikum kali ini:

1. Buat package dengan nama **pekan6**
2. Buat file class baru dalam package, dengan nama **NodeDLL**
3. Buat class NodeDLL, deklarasikan data (integer), next (NodeDLL), prev (NodeDLL)

```
3 public class NodeDLL {
4     int data;
5     NodeDLL next;
6     NodeDLL prev;
```

4. Konstruktor NodeDLL diisi dengan this.data, this.next, dan this.prev

```
8 public NodeDLL(int data) {
9     this.data = data;
10    this.next = null;
11    this.prev = null;
12 }
13
14 // Jovantri Immanuel Gulo
15 // 2411532014
16 }
```

5. Buat file class baru lagi dengan nama **InsertDLL**
6. Buat public class untuk InsertDLL
7. Method static insertBegin dengan query head dan newData. Bagian ini memproses node yang kita input, menjadi node yang pertama, jika ada node pertama, akan menjadi node kedua, karena kita input at begin.

```
3 public class InsertDLL {
4     static NodeDLL insertBegin(NodeDLL head, int data) {
5         NodeDLL new_node = new NodeDLL(data);
6         new_node.next = head;
7         if(head != null) {
8             head.prev = new_node;
9         }
10        return new_node;
11    }
```

8. Method static insertEnd dengan query head dan newData. Pada bagian ini kita menggunakan **curr** untuk mengidentifikasi apakah node sudah diperiksa hingga urutan akhir node, sehingga pada saat diperiksa hingga baris akhir, node terakhir akan menjadi newData tadi, sehingga node akan tertambahkan pada bagian akhir node.

```
13 public static NodeDLL insertEnd(NodeDLL head, int newData) {
14     NodeDLL newNode = new NodeDLL(newData);
15     if(head == null) {
16         head = newNode;
17     } else {
18         NodeDLL curr = head;
19         while(curr.next != null) {
20             curr = curr.next;
21         }
22         curr.next = newNode;
23         newNode.prev = curr;
24     }
25     return head;
26 }
```

9. Method insertAtPosition dengan query head, pos, dan new_data. Pos merupakan posisi node spesifik yang ingin kita selipkan.
- Otomatis menjadikan headnya sebagai new_node apabila posnya adalah 1.
 - Apabila head tidak null, maka sebelum head, akan diset menjadi new_node (sebelum head artinya pertama, karena pos nya 1).
 - Curr akan terus menelusuri sampai ke posisi sebelum pos yang dituju (pos - 1).
 - Kalau posisi tidak valid (curr == null), maka akan muncul pesan “Posisi ini tidak ada”.
 - Menyisipkan new_node, sehingga tepat pada posisi pos yang dituju.

```
28 public static NodeDLL insertAtPosition(NodeDLL head, int pos, int new_data) {
29     NodeDLL new_node = new NodeDLL(new_data);
30     if(pos == 1) {
31         new_node.next = head;
32         if(head != null) {
33             head.prev = new_node;
34         }
35         head = new_node;
36         return head;
37     }
38     NodeDLL curr = head;
39     for(int i = 1; i < pos - 1 && curr != null; i++) {
40         curr = curr.next;
41     }
42     if(curr == null) {
43         System.out.println("Posisi tidak ada.");
44         return head;
45     }
46     new_node.prev = curr;
47     new_node.next = curr.next;
48     curr.next = new_node;
49     if(new_node.next != null) {
50         new_node.next.prev = new_node;
51     }
52     return head;
53 }
```

10. Method `printList` akan membantu kita dalam menampilkan seluruh double linked list yang telah dilakukan konfigurasi tadi, dengan perulangan `while`.

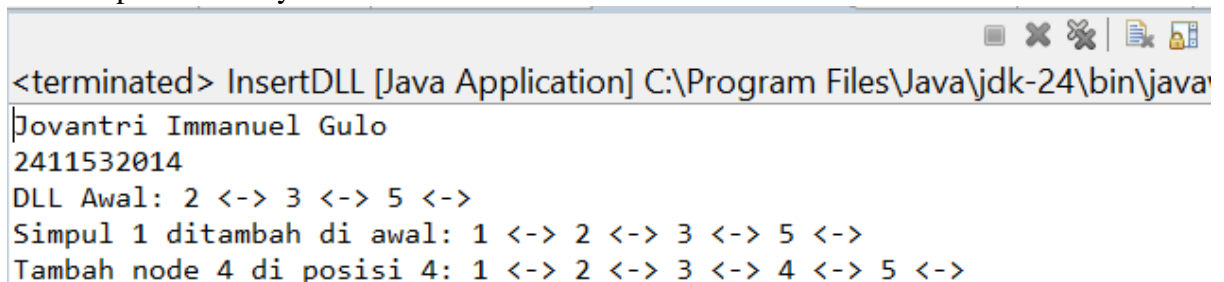
```
55 public static void printList(NodeDLL head) {
56     NodeDLL curr = head;
57     while(curr != null) {
58         System.out.print(curr.data + " ↔ ");
59         curr = curr.next;
60     }
61     System.out.println();
62 }
```

11. Pada kelas `main`, lakukan deklarasi node head baru dengan `new NodeDLL(2)`, kemudian deklarasikan node ke dua yaitu `new NodeDLL(3)`, dan node ke tiga yaitu `new NodeDLL(5)`.

- Menampilkan senarai berantai DLL awal
- Meletakkan simpul 1 pada bagian awal, lalu ditampilkan
- Menambah node 4 di posisi ke 4, lalu ditampilkan

```
64 public static void main(String[] args) {
65     System.out.println("Jovantri Immanuel Gulo");
66     System.out.println("2411532014");
67
68     NodeDLL head = new NodeDLL(2);
69     head.next = new NodeDLL(3);
70     head.next.prev = head;
71     head.next.next = new NodeDLL(5);
72     head.next.next.prev = head.next;
73
74     System.out.print("DLL Awal: ");
75     printList(head);
76
77     head = insertBegin(head, 1);
78     System.out.print("Simpul 1 ditambah di awal: ");
79     printList(head);
80
81     System.out.print("Tambah node 4 di posisi 4: ");
82     int data2 = 4;
83     int pos = 4;
84     head = insertAtPosition(head, pos, data2);
85     printList(head);
86 }
87 }
```

12. Ini merupakan hasilnya



```
<terminated> InsertDLL [Java Application] C:\Program Files\Java\jdk-24\bin\java
Jovantri Immanuel Gulo
2411532014
DLL Awal: 2 <-> 3 <-> 5 <->
Simpul 1 ditambah di awal: 1 <-> 2 <-> 3 <-> 5 <->
Tambah node 4 di posisi 4: 1 <-> 2 <-> 3 <-> 4 <-> 5 <->
```

13. Buat file class baru dengan nama **HapusDLL**

14. Buat public class nya dan juga method static delHead dengan query head (tipe data NodeDLL).

- Apabila head null, maka return null juga
- Deklarasi temp sebagai head
- Kemudian, deklarasi head sebagai head.next
- Apabila head tidak null (ada nilai), maka set head.prev sama dengan null
- Kembalikan head

```
3 public class HapusDLL {
4     public static NodeDLL delHead(NodeDLL head) {
5         if(head == null) {
6             return null;
7         }
8         NodeDLL temp = head;
9         head = head.next;
10        if(head != null) {
11            head.prev = null;
12        }
13        return head;
14    }
```

15. Buat method static delLast dengan query head juga.

- Apabila head null, maka return null juga
- Apabila head.next (node ke 2) null juga, maka return null juga
- Deklarasikan objek curr sebagai head
- Dengan perulangan while, (apabila curr tidak null adalah true) maka set curr = curr.next, yaitu memeriksa seluruh node, hingga node akhir.
- Kemudian set curr.prev.next = null, supaya pointer yang paling terakhir dapat terhapus.
- Mengembalikan head

```
16     public static NodeDLL delLast(NodeDLL head) {
17         if(head == null) {
18             return null;
19         }
20         if(head.next == null) {
21             return null;
22         }
23         NodeDLL curr = head;
24         while(curr.next != null) {
25             curr = curr.next;
26         }
27         if(curr.prev != null) {
28             curr.prev.next = null;
29         }
30         return head;
31     }
```

16. Buat method static delPos dengan query head dan pos (int).

- Apabila head nya null, maka return null
- Deklarasikan objek curr sebagai head
- Mencari node pada posisi pos yang diinginkan
- Return head apabila posisinya melebihi panjang list

- e. Memutuskan hubungan dengan node yang sebelumnya dan juga memutuskan hubungan dengan node yang setelahnya
- f. Apabila yang dihapus adalah head, maka ganti dengan head yang baru, yaitu curr.next
- g. Return head yang baru

```

33 public static NodeDLL delPos(NodeDLL head, int pos) {
34     if(head == null) {
35         return head;
36     }
37     NodeDLL curr = head;
38     for(int i = 1; curr != null && i < pos; ++i) {
39         curr = curr.next;
40     }
41     if(curr == null) {
42         return head;
43     }
44     if(curr.prev != null) {
45         curr.prev.next = curr.next;
46     }
47     if(curr.next != null) {
48         curr.next.prev = curr.prev;
49     }
50     if(head == curr) {
51         head = curr.next;
52     }
53     return head;
54 }

```

17. Buat method printList untuk menampilkan seluruh node linked list yang telah dikonfigurasi.

```

56 public static void printList(NodeDLL head) {
57     NodeDLL curr = head;
58     while(curr != null) {
59         System.out.print(curr.data + " ");
60         curr = curr.next;
61     }
62     System.out.println();
63 }

```

18. Pada kelas main,

- a. kita membuat node satu per satu yaitu 1 – 2 – 3 – 4 – 5 dengan bolak-balik yaitu keduanya saling terhubung (sebelum dan sesudah)
- b. Menampilkan list di awal
- c. Menghapus node pertama dengan menggunakan delHead, lalu ditampilkan
- d. Menghapus node ke 2 dengan menggunakan delPos, lalu ditampilkan

```

65 public static void main(String[] args) {
66     System.out.println("Jovantri Immanuel Gulo");
67     System.out.println("2411532014");
68     System.out.println();
69
70     NodeDLL head = new NodeDLL(1);
71     head.next = new NodeDLL(2);
72     head.next.prev = head;
73     head.next.next = new NodeDLL(3);
74     head.next.next.prev = head.next;
75     head.next.next.next = new NodeDLL(4);
76     head.next.next.next.prev = head.next.next;
77     head.next.next.next.next = new NodeDLL(5);
78     head.next.next.next.next.prev = head.next.next.next;
79
80     System.out.print("DLL Awal: ");
81     printList(head);
82
83     System.out.print("Setelah head dihapus: ");
84     head = delHead(head);
85     printList(head);
86
87     System.out.print("Menghapus node ke-2: ");
88     head = delPos(head, 2);
89     printList(head);

```

19. Ini merupakan hasilnya

```

<terminated> HapusDLL [Java Application] C:\Progra
Jovantri Immanuel Gulo
2411532014

DLL Awal: 1 2 3 4 5
Setelah head dihapus: 2 3 4 5
Menghapus node ke-2: 2 4 5

```

20. Buat file class baru dengan nama **PenelusuranDLL**

21. Buat public class nya dan method static forwardTraversal dengan query NodeDLL head.

- a. Deklarasikan curr sebagai head node
- b. Perulangan while saat curr tidak null
- c. Menampilkan seluruh nilai node yang ada pada curr

```

3 public class PenelusuranDLL {
4     static void forwardTraversal(NodeDLL head) {
5         NodeDLL curr = head;
6
7         while(curr ≠ null) {
8             System.out.print(curr.data + " ↔ ");
9             curr = curr.next;
10        }
11        System.out.println();
12    }

```

22. Method static backwardTraversal dengan query NodeDLL tail.

- Deklarasikan curr node sebagai tail
- Menggunakan perulangan while untuk menampilkan seluruh data node yang ada pada linked list
- Menggunakan curr.prev dikarenakan diurutkan secara mundur
- Menampilkan seluruh data dengan posisi terbalik, yaitu kebalikan dari forwardTraversal

```

14 static void backwardTraversal(NodeDLL tail) {
15     NodeDLL curr = tail;
16
17     while(curr ≠ null) {
18         System.out.print(curr.data + " ↔ ");
19         curr = curr.prev;
20     }
21     System.out.println();
22 }

```

23. Kemudian pada kelas main.

- Deklarasikan head dengan NodeDLL 1, second dengan NodeDLL 2, dan third dengan NodeDLL 3.
- Melakukan penghubungan pada tiap-tiap node, sehingga tiap nodenya saling terhubung satu sama lain dengan posisi dan urutannya
- Menampilkan penelusuran maju dengan menggunakan *forwardTraversal* yang dimulai dari first node
- Menampilkan penelusuran mundur dengan menggunakan *backwardTraversal* yang dimulai dari third node


```

24 public static void main(String[] args) {
25     System.out.println("Jovantri Immanuel Gulo");
26     System.out.println("2411532014");
27     System.out.println();
28
29     NodeDLL head = new NodeDLL(1);
30     NodeDLL second = new NodeDLL(2);
31     NodeDLL third = new NodeDLL(3);
32
33     head.next = second;
34     second.prev = head;
35     second.next = third;
36     third.prev = second;
37
38     System.out.println("Penelusuran maju: ");
39     forwardTraversal(head);
40
41     System.out.println("Penelusuran mundur: ");
42     backwardTraversal(third);
43 }

```

24. Ini adalah hasilnya

```

<terminated> PenelusuranDLL [Java Application] C:\Program Files
Jovantri Immanuel Gulo
2411532014

Penelusuran maju:
1 <-> 2 <-> 3 <->
Penelusuran mundur:
3 <-> 2 <-> 1 <->

```

D. Kesimpulan

Dari praktikum yang telah dilakukan, dapat diambil kesimpulan bahwa doubly linked list menawarkan efisiensi dalam manipulasi data, terutama dalam konteks penghapusan dan penyisipan elemen di berbagai posisi dalam struktur. Kemampuan traversal dua arah memberikan kemudahan dalam mengakses data, meskipun kompleksitas implementasinya lebih tinggi dibandingkan singly linked list. Dengan memahami konsep ini, para pengguna dapat lebih optimal dalam memilih struktur data yang sesuai dengan kebutuhan aplikasi yang sedang dikembangkan.