FTI

TEKNIK ELEKTRO

EIE5023

ALGORITMA DAN STRUKTUR DATA + PRAK

3

ENDANG SRI RAHAYU

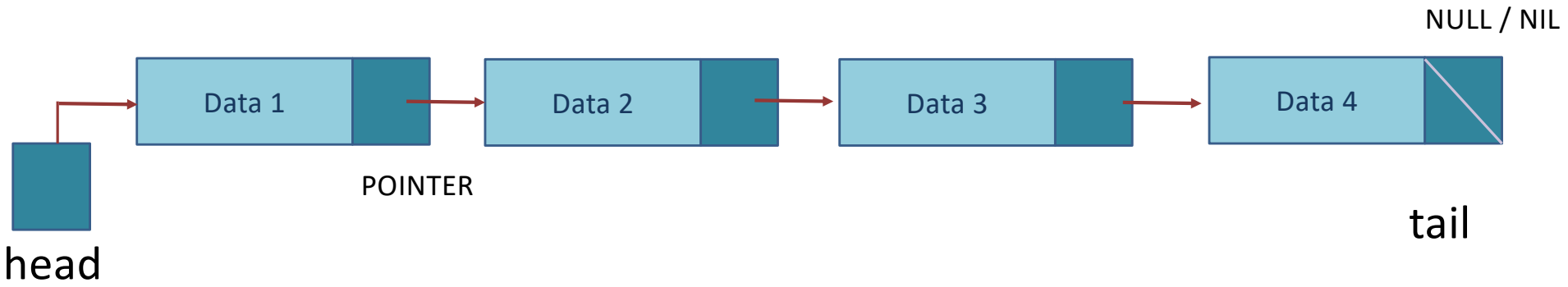
Outlines:

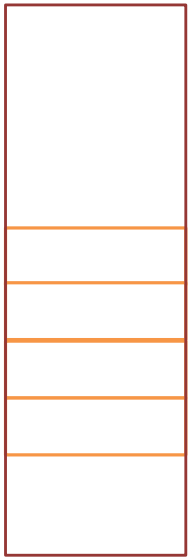
Linked List:

- Pengertian Struktur Data Linked List
- Operasi pada Linked List (penambahan, penghapusan) data di awal, tengah dan akhir list,
- Kelebihan, kekurangan Linked list
- Praktik: Pemrograman Python

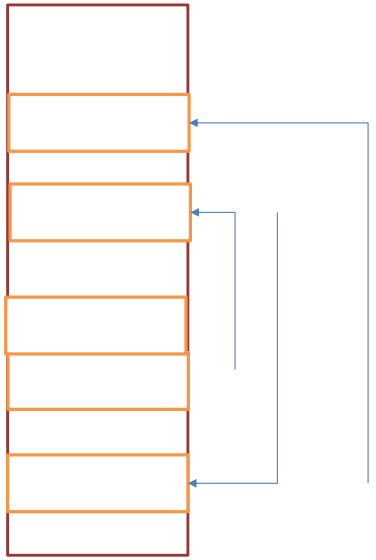
Linked List:

- salah satu struktur data dalam pemrograman yang digunakan untuk menyimpan dan mengelola data secara dinamis.
- Dapat dengan mudah membuat tempat baru untuk menyimpan data kapan saja dibutuhkan
- data disimpan dalam bentuk simpul atau node yang saling terhubung satu sama lain





ARRAY

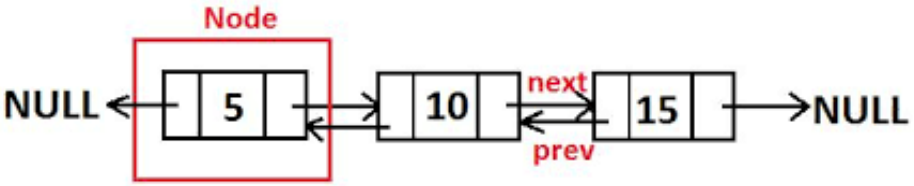


LINKED LIST

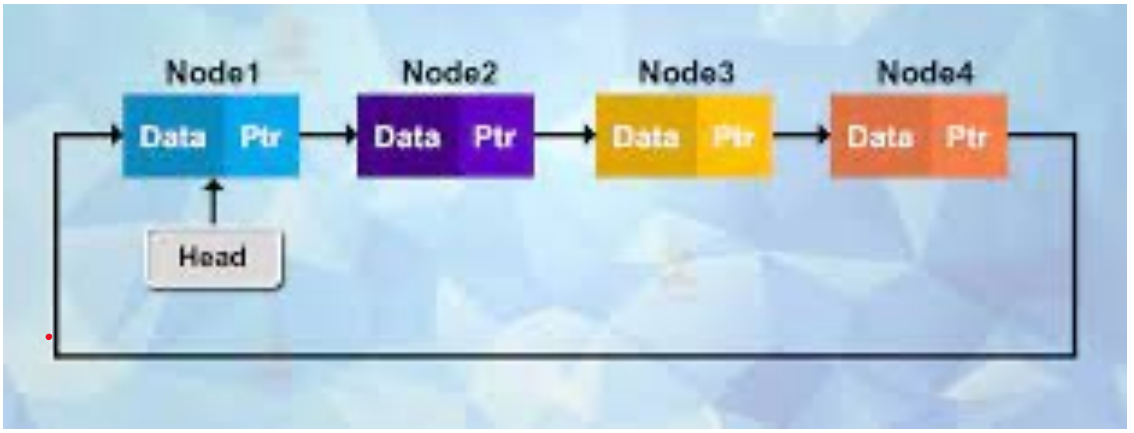
Fungsi:

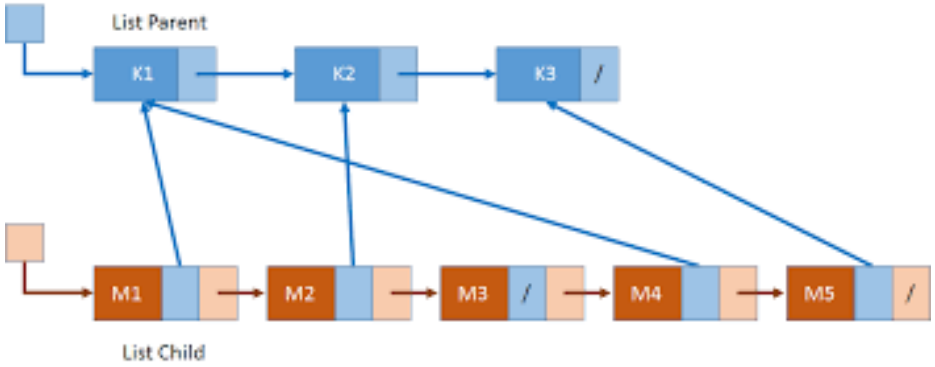
- Menyimpan dan mengelola data dalam urutan tertentu
- Memudahkan penambahan dan penghapusan data secara dinamis tanpa harus menggeser data lain
- Digunakan dalam implementasi berbagai algoritma dan struktur data lain seperti stack dan queue

Jenis:

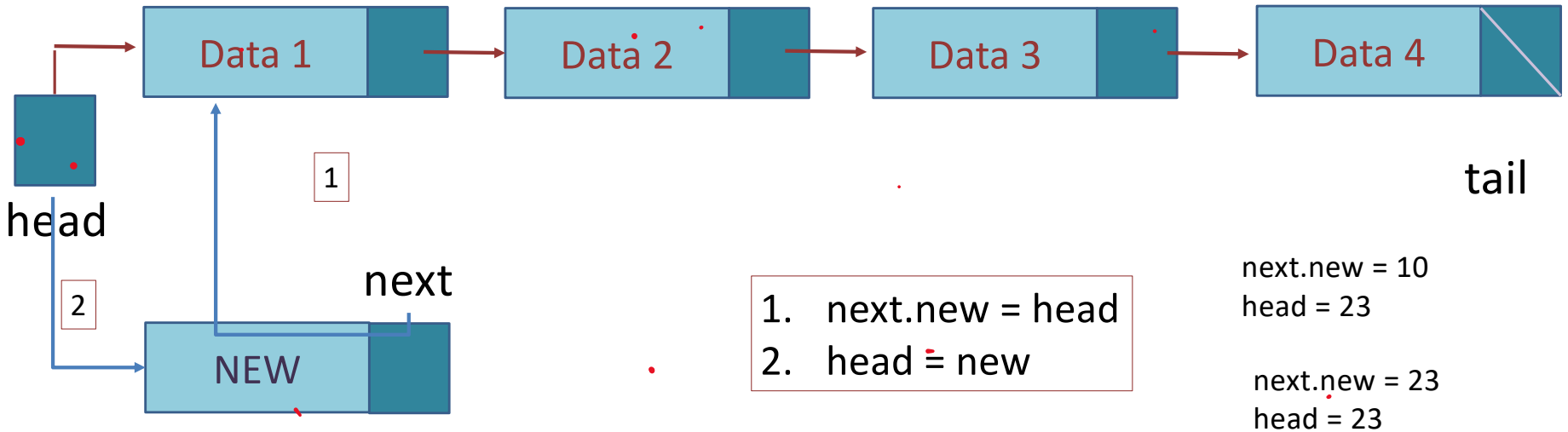


- Single Linked List
- Double Linked List
- Circular Linked List
- Multiple Linked List





ALGORITMA OPERASI: insert didepan,



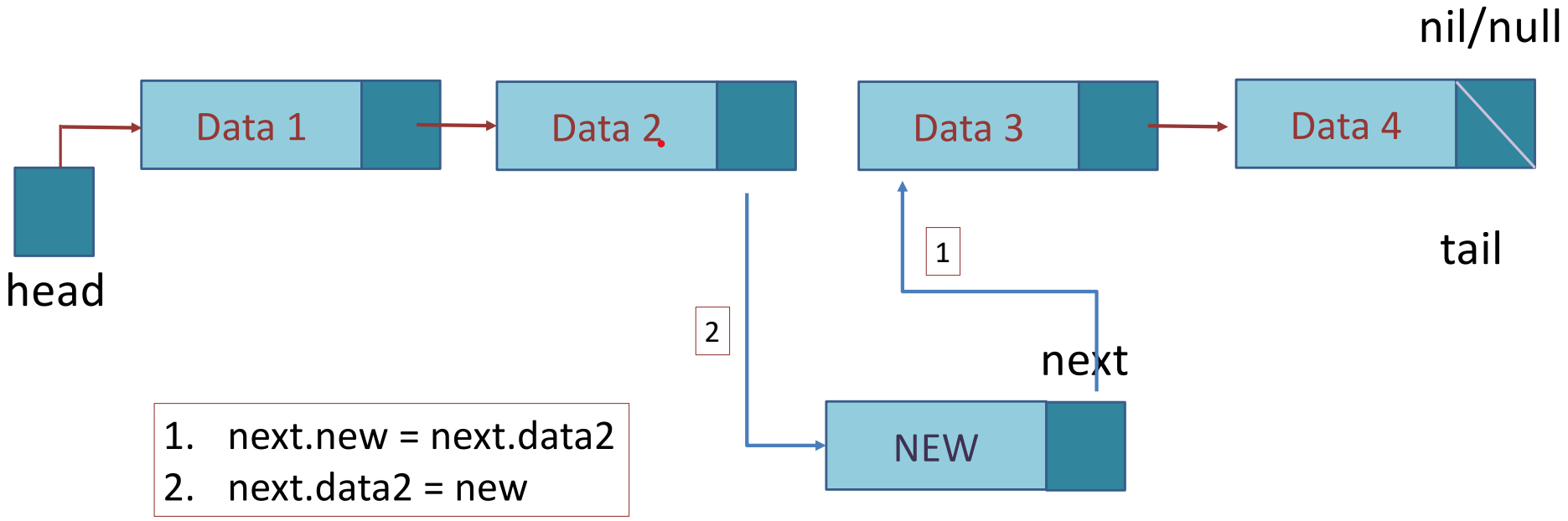
1. next.new = head
2. head = new

next.new = 10
head = 23

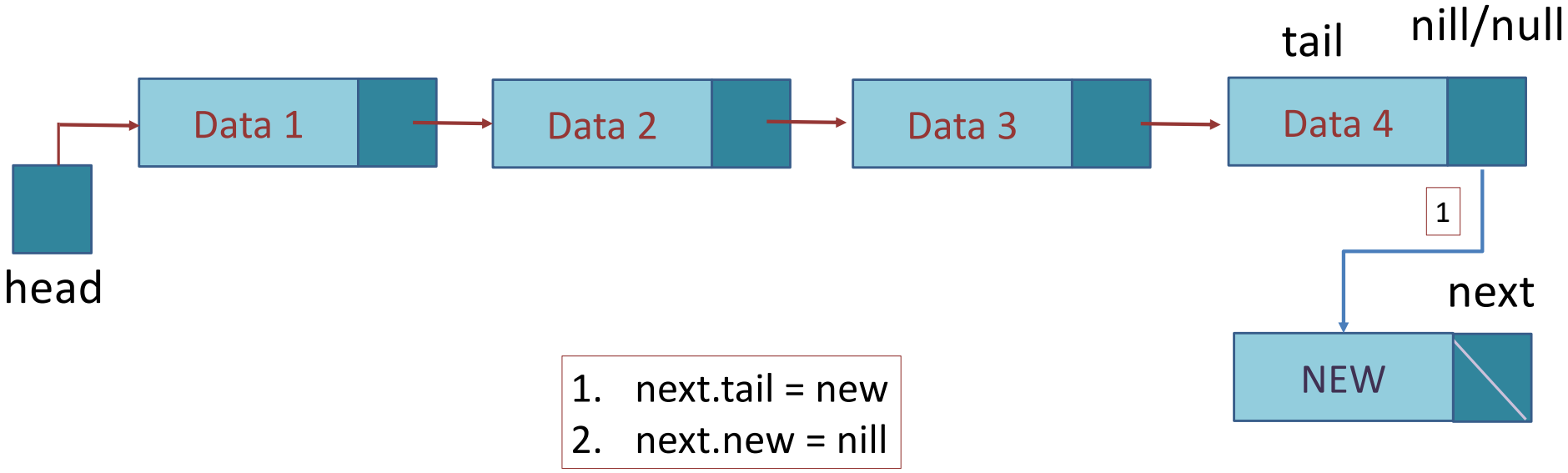
next.new = 23
head = 23

head = 65

ALGORITMA OPERASI: insert ditengah,

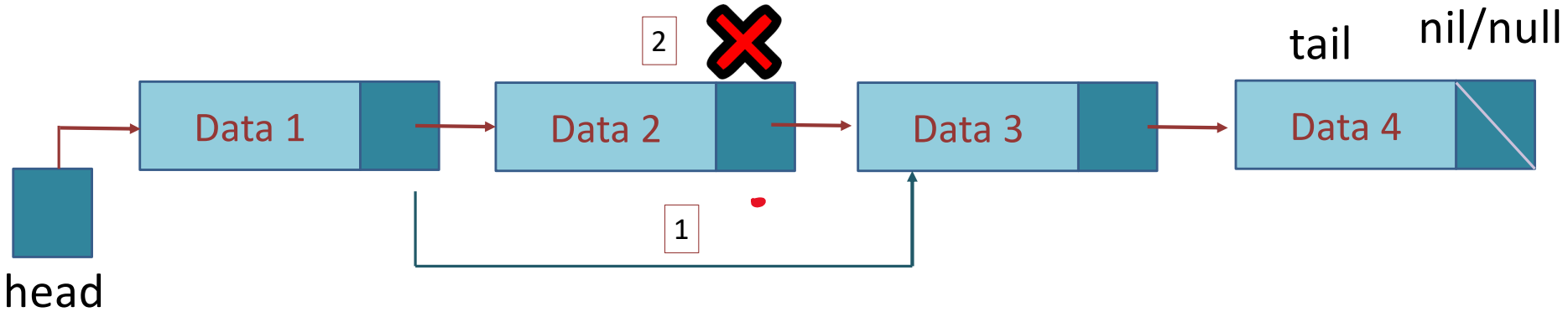


ALGORITMA OPERASI: insert di belakang



1. next.tail = new
2. next.new = nill

ALGORITMA OPERASI: delete di tengah



1. next.data1 = next.data2
2. Delete(Data2)

DISKUSI

- Array memerlukan alokasi memori kontiguitas, yang berarti elemen data harus disusun secara berurutan dalam memori. Ini membuat penambahan atau penghapusan elemen menjadi lebih sulit, karena kita perlu menggeser elemen lain untuk memberi ruang pada posisi yang diinginkan

- ukuran Array tetap dan ditentukan saat deklarasi, sehingga sulit untuk mengubah ukuran Array secara dinamis.
- Ukuran Linked List dapat berubah selama runtime
- Array memberikan akses elemen yang cepat melalui indeks
- Linked List merupakan pilihan yang fleksibel untuk situasi di mana data sering berubah atau ketika ukuran data tidak diketahui sebelumnya

```
class Node:
"""Kelas untuk merepresentasikan node dalam linked
list."""
def __init__(self, data):
self.data = data
self.next = None

class LinkedList:
"""Kelas untuk mengelola operasi pada linked list."""
def __init__(self):
self.head = None

def append(self, data):
"""Menambahkan node di akhir linked list."""
new_node = Node(data)
if not self.head:
self.head = new_node
return
last = self.head
while last.next:
last = last.next
last.next = new_node
```

```
def prepend(self, data):
"""Menambahkan node di awal linked list."""
new_node = Node(data)
new_node.next = self.head
self.head = new_node

def delete_node(self, key):
"""Menghapus node berdasarkan nilai
tertentu."""
temp = self.head

if temp and temp.data == key:
self.head = temp.next
temp = None
return

prev = None
while temp and temp.data != key:
prev = temp
temp = temp.next
```

Prakt: Program Python (1) – Linked List

```python
class Node:
    """Kelas untuk merepresentasikan node dalam linked list."""
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    """Kelas untuk mengelola operasi pada linked list."""
    def __init__(self):
        self.head = None

    def append(self, data):
        """Menambahkan node di akhir linked list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
        last = self.head
        while last.next:
            last = last.next
        last.next = new_node

    def prepend(self, data):
        """Menambahkan node di awal linked list."""
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node
```

```
    def delete_node(self, key):
        """Menghapus node berdasarkan nilai tertentu."""
        temp = self.head

        if temp and temp.data == key:
            self.head = temp.next
            temp = None
            return

        prev = None
        while temp and temp.data != key:
            prev = temp
            temp = temp.next

        if temp is None:
            return

        prev.next = temp.next
        temp = None

    def display(self):
        """Menampilkan isi linked list."""
        temp = self.head
        while temp:
            print(temp.data, end=" -> ")
            temp = temp.next
        print("None")
```

```python
# Contoh penggunaan linked list
llist = LinkedList()
llist.append(10)
llist.append(20)
llist.append(30)
llist.prepend(5)

print("Linked List setelah penambahan:")
llist.display()

llist.delete_node(20)
print("Linked List setelah menghapus node dengan nilai 20:")
llist.display()
```

Output:

```
Linked List setelah penambahan:
5 -> 10 -> 20 -> 30 -> None
Linked List setelah menghapus node dengan nilai 20:
5 -> 10 -> 30 -> None
```

Penjelasan Program:

- **Node**: Setiap node memiliki **data** dan **pointer** ke node berikutnya.
- **LinkedList**: Mengelola operasi seperti **append()**, **prepend()**, **delete_node()**, dan **display()**.
- **append()**: Menambahkan elemen di akhir linked list.
- **prepend()**: Menambahkan elemen di awal linked list.
- **delete_node()**: Menghapus elemen dengan nilai tertentu.
- **display()**: Menampilkan elemen-elemen dalam linked list.

Prakt: Program Python (2) – Doubly Linked List

```python
class Node:
    """Kelas untuk merepresentasikan node dalam Doubly Linked List."""
    def __init__(self, data):
        self.data = data
        self.next = None  # Pointer ke node berikutnya
        self.prev = None  # Pointer ke node sebelumnya

class DoublyLinkedList:
    """Kelas untuk mengelola operasi pada Doubly Linked List."""
    def __init__(self):
        self.head = None

    def append(self, data):
        """Menambahkan node di akhir linked list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return

        last = self.head
        while last.next:
            last = last.next
        last.next = new_node
        new_node.prev = last

    def prepend(self, data):
        """Menambahkan node di awal linked list."""
        new_node = Node(data)
        if not self.head:
            self.head = new_node
            return
```

```
        new_node.next = self.head
        self.head.prev = new_node
        self.head = new_node

    def delete_node(self, key):
        """Menghapus node berdasarkan nilai tertentu."""
        temp = self.head

        # Jika node yang ingin dihapus adalah head
        if temp and temp.data == key:
            if temp.next:
                temp.next.prev = None
            self.head = temp.next
            temp = None
            return

        while temp and temp.data != key:
            temp = temp.next

        if temp is None:
            return  # Jika node tidak ditemukan

        if temp.next:
            temp.next.prev = temp.prev
        if temp.prev:
            temp.prev.next = temp.next

        temp = None
```

```python
    def display_forward(self):
        """Menampilkan isi linked list dari depan ke belakang."""
        temp = self.head
        while temp:
            print(temp.data, end=" ⇄ ")
            temp = temp.next
        print("None")

    def display_backward(self):
        """Menampilkan isi linked list dari belakang ke depan."""
        temp = self.head
        if not temp:
            return

        while temp.next:
            temp = temp.next

        while temp:
            print(temp.data, end=" ⇄ ")
            temp = temp.prev
        print("None")
```

```python
# Contoh penggunaan Doubly Linked List
dll = DoublyLinkedList()
dll.append(10)
dll.append(20)
dll.append(30)
dll.prepend(5)

print("Doubly Linked List (maju):")
dll.display_forward()

print("Doubly Linked List (mundur):")
dll.display_backward()

dll.delete_node(20)
print("Setelah menghapus node dengan nilai 20:")
dll.display_forward()
```

Output:

```
Doubly Linked List (maju):
5 ⇄ 10 ⇄ 20 ⇄ 30 ⇄ None
Doubly Linked List (mundur):
30 ⇄ 20 ⇄ 10 ⇄ 5 ⇄ None
Setelah menghapus node dengan nilai 20:
5 ⇄ 10 ⇄ 30 ⇄ None
```

Penjelasan Program:

1.Node dalam Doubly Linked List

1.Setiap node memiliki **data**, **pointer ke node berikutnya (next)**, dan **pointer ke node sebelumnya (prev)**.

2.Operasi yang didukung

1. **append(data)**: Menambahkan node di akhir linked list.
2. **prepend(data)**: Menambahkan node di awal linked list.
3. **delete_node(key)**: Menghapus node dengan nilai tertentu.
4. **display_forward()**: Menampilkan isi linked list dari depan ke belakang.
5. **display_backward()**: Menampilkan isi linked list dari belakang ke depan.

TERIMA KASIH

Next ---- PERTEMUAN ke-4

TEKNIK ELEKTRO
FTI UJ

ENDANG SRI RAHAYU