```
class Node:
    def __init__(self, data):
        self.data = data
        self.prev = None
        self.next = None
```

Kelas Node: Mewakili elemen dari linked list.

__init__: Konstruktor yang menginisialisasi node dengan:

data: Nilai yang disimpan dalam node.

prev: Pointer ke node sebelumnya (diinisialisasi dengan None).

next: Pointer ke node berikutnya (diinisialisasi dengan None).

```
class DoubleLinkedList:
    def __init__(self):
        self.head = None
```

Kelas DoubleLinkedList: Mewakili linked list itu sendiri.

__init__: Konstruktor yang menginisialisasi linked list dengan:

head: Pointer ke node pertama (diinisialisasi dengan None).

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

append: Menambahkan node baru di akhir linked list.

Membuat new_node dengan data yang diberikan.

Jika head kosong, new_node menjadi head.

Jika tidak, iterasi dilakukan untuk menemukan node terakhir, lalu new_node ditambahkan setelahnya dan pointer prev diatur.

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

display: Menampilkan semua elemen dalam linked list.

Menggunakan loop untuk mencetak data dari setiap node hingga mencapai akhir (node None).

```
def delete_awal(self):
    """Menghapus node awal"""
    if not self.head:
        print("Linked list kosong!")
        return
    if not self.head.next:
        self.head = None
    else:
        self.head = self.head.next
        self.head.prev = None
```

delete_awal: Menghapus node pertama dari linked list.

Jika linked list kosong, cetak pesan.

Jika hanya ada satu node, set head ke None.

Jika ada lebih dari satu node, atur head ke node berikutnya dan hapus pointer prev.

```
def delete_akhir(self):
    """Menghapus node akhir"""
    if not self.head:
        print("Linked list kosong!")
```

```
        return
    curr = self.head
    if not curr.next:
        self.head = None
        return
    while curr.next:
        curr = curr.next
    curr.prev.next = None
```

delete_akhir: Menghapus node terakhir dari linked list.

Jika linked list kosong, cetak pesan.

Jika hanya ada satu node, set head ke None.

Jika ada lebih dari satu node, iterasi dilakukan untuk menemukan node terakhir dan menghapusnya dengan mengatur pointer next dari node sebelumnya ke None.

```
def delete_berdasarkan_nilai(self, target):
    """Menghapus node berdasarkan nilai data"""
    if not self.head:
        print("Linked list kosong!")
        return

    curr = self.head

    # Jika node yang akan dihapus adalah head
    if curr.data == target:
        self.delete_awal()
        return

    while curr:
        if curr.data == target:
            if curr.next:
                curr.prev.next = curr.next
                curr.next.prev = curr.prev
            else:
                # Node terakhir
                curr.prev.next = None
```

```
            return
        curr = curr.next
    print(f"Data {target} tidak ditemukan dalam linked list.")
```

delete_berdasarkan_nilai: Menghapus node berdasarkan nilai yang diberikan.

Jika linked list kosong, cetak pesan.

Jika node yang akan dihapus adalah head, panggil delete_awal.

Jika tidak, iterasi dilakukan untuk menemukan node dengan nilai yang sesuai dan menghapusnya dengan mengatur pointer next dan prev yang sesuai.

Jika tidak ditemukan, cetak pesan bahwa data tidak ada.