# LAPORAN PRAKTIKUM
# PERTEMUAN 6
# DOUBLE LINKED LIST BAGIAN 1


Nama :

Viona Aziz Syahputri (2311104008)

Dosen :

YUDHA ISLAMI SULISTYA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

**TELKOM UNIVERSITY PURWOKERTO**


**2024**

1. **Soal**

```cpp
#include <iostream>

using namespace std;

// Struktur Node untuk menyimpan data
struct Node {
    int data;
    Node* prev;
    Node* next;
};

// Kelas untuk mengelola Doubly Linked List
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    // Konstruktor
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Fungsi untuk menambahkan elemen di awal list
    void insertFirst_2311104008(int value) {
        Node* newNode = new Node{value, nullptr, nullptr};
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    // Fungsi untuk menambahkan elemen di akhir list
    void insertLast_2311104008(int value) {
        Node* newNode = new Node{value, nullptr, nullptr};
        if (tail == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
```

```
        }
    }


    void display_2311104008() {
        Node* current = head;
        cout << "DAFTAR ANGGOTA LIST: ";
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << " <-> ";
            }
            current = current->next;
        }
        cout << endl;
    }

    Node* searchNode_2311104008(int value) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == value) {
                return current;
            }
            current = current->next;
        }
        return nullptr;
    }

    ~DoublyLinkedList() {
        Node* current = head;
        while (current != nullptr) {
            Node* nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};
int main() {
    DoublyLinkedList dll;

    // Input dari pengguna
    int elemenPertama, elemenKedua, elemenKetiga;
    cout << "Masukkan elemen pertama = ";
    cin >> elemenPertama;
    dll.insertLast_2311104008(elemenPertama);

    cout << "Masukkan elemen kedua di awal = ";
    cin >> elemenKedua;
    dll.insertFirst_2311104008(elemenKedua);

    cout << "Masukkan elemen ketiga di akhir = ";
    cin >> elemenKetiga;
    dll.insertLast_2311104008(elemenKetiga);

    // Menampilkan seluruh elemen dalam list
    dll.display_2311104008();

    return 0;
}
```

## Output

```
Masukkan elemen pertama = 10
Masukkan elemen kedua di awal = 5
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST: 5 <-> 10 <-> 20
```

Dalam blok kode ini, terdapat implementasi dari Doubly Linked List yang terdiri dari struktur Node yang menyimpan data dan pointer ke node sebelumnya, disebut prev, dan node selanjutnya, disebut next sehingga traversal dua arah dapat dilakukan. Selain itu, terdapat kelas DoublyLinkedList untuk mengelola linked list dengan dua pointer, yakni head dan tail, serta konstruktor yang menginisialisasi kedua pointer tersebut dengan nullptr untuk menandakan list kosong, terdapat fungsi insertFirst_2311104008() untuk menambahkan elemen baru pada awal linked list dimana node baru menjadi head dan tail jika list kosong atau dihubungkan ke node pertama jika tidak, fungsi insertLast_2311104008() untuk menambahkan elemen baru pada akhir linked list pada cara yang sama, fungsi display_2311104008() yang menampilkan semua elemen dalam linked list dengan format yang menunjukkan hubungan antar node, fungsi searchNode_2311104008() yang mencari node dengan nilai yang diberikan dan memberi pointer ke node jika ditemukan atau nullptr jika tidak, destruktor yang menghapus semua node dalam linked list, fungsi main yang membuat objek DoublyLinkedList, mengambil input pengguna untuk menambahkan elemen pada linked list dan melakukan pemanggilan display() pada fungsi tersebut.

2. SOAL

```cpp
#include <iostream>

using namespace std;

// Struktur Node untuk menyimpan data
struct Node {
    int data;
    Node* prev;
    Node* next;
};

// Kelas untuk mengelola Doubly Linked List
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Fungsi untuk menambahkan elemen di akhir list
    void insertLast_2311104008(int value) {
        Node* newNode = new Node{value, nullptr, nullptr};
        if (tail == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    // Fungsi untuk menghapus elemen pertama
    void deleteFirst_2311104008() {
        if (head == nullptr) {
            cout << "List kosong, tidak ada elemen yang dapat dihapus." << endl;
            return;
        }
```

```
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    } else {
        tail = nullptr;
    }
    delete temp;
}

// Fungsi untuk menghapus elemen terakhir
void deleteLast_2311104008() {
    if (tail == nullptr) {
        cout << "List kosong, tidak ada elemen yang dapat dihapus." << endl;
        return;
    }
    Node* temp = tail;
    tail = tail->prev;
    if (tail != nullptr) {
        tail->next = nullptr;
    } else {
        head = nullptr;
    }
    delete temp;
}

// Fungsi untuk menampilkan seluruh elemen dalam list dari depan ke belakang
void display_2311104008() {
    Node* current = head;
    cout << "";
    while (current != nullptr) {
        cout << current->data;
        if (current->next != nullptr) {
            cout << " <-> ";
        }
        current = current->next;
    }
    cout << endl;
}
    // Destructor untuk membersihkan memori
    ~DoublyLinkedList() {
        Node* current = head;
        while (current != nullptr) {
            Node* nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main() {
    DoublyLinkedList dll;

    // Input dari pengguna
    int elemenPertama, elemenKedua, elemenKetiga;
    cout << "Masukkan elemen pertama = ";
    cin >> elemenPertama;
    dll.insertLast_2311104008(elemenPertama);

    cout << "Masukkan elemen kedua di akhir = ";
    cin >> elemenKedua;
    dll.insertLast_2311104008(elemenKedua);

    cout << "Masukkan elemen ketiga di akhir = ";
    cin >> elemenKetiga;
    dll.insertLast_2311104008(elemenKetiga);

    // Menghapus elemen pertama dan terakhir
    dll.deleteFirst_2311104008();
    dll.deleteLast_2311104008();

    // Menampilkan seluruh elemen dalam list setelah penghapusan
    cout << "DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: ";
    dll.display_2311104008();

    return 0;
}
```

Output

```
Masukkan elemen pertama = 10
Masukkan elemen kedua di akhir = 15
Masukkan elemen ketiga di akhir = 20
DAFTAR ANGGOTA LIST SETELAH PENGHAPUSAN: 15
```

Kodingan ini mengimplementasikan Doubly Linked List dengan struktur Node yang menyimpan data integer dan dua pointer (prev dan next) untuk traversal dua arah. Kelas DoublyLinkedList mengelola list dengan pointer head dan tail, diinisialisasi dengan nullptr untuk menandakan list kosong. Fungsi **insertLast_2311104008** menambahkan Elemen di akhir list sementara **deleteFirst_2311104008** dan **deleteLast_2311104008** menghapus elemen pertama dan terakhir, masing-masing, dengan penanganan untuk list kosong. Fungsi **display_2311104008** menampilkan semua elemen dari depan ke belakang, dan destruktor membersihkan semua node untuk mencegah kebocoran memori. Dalam fungsi main, objek DoublyLinkedList dibuat, tiga elemen ditambahkan di akhir list berdasarkan input pengguna, kemudian elemen pertama dan terakhir dihapus, dan seluruh elemen ditampilkan setelah penghapusan.

3. SOAL

```cpp
#include <iostream>

using namespace std;

// Struktur Node untuk menyimpan data
struct Node {
    int data;
    Node* prev;
    Node* next;
};

// Kelas untuk mengelola Doubly Linked List
class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Fungsi untuk menambahkan elemen di akhir list
    void insertLast_2311104008(int value) {
        Node* newNode = new Node{value, nullptr, nullptr};
        if (tail == nullptr) {
            tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    // Fungsi untuk menampilkan elemen dari depan ke belakang
    void displayForward_2311104008() {
        Node* current = head;
        cout << "Daftar elemen dari depan ke belakang: ";
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << " <-> ";
            }
            current = current->next;
        }
        cout << endl;
```

```
    // Fungsi untuk menampilkan elemen dari belakang ke depan
    void displayBackward_2311104008() {
        Node* current = tail;
        cout << "Daftar elemen dari belakang ke depan: ";
        while (current != nullptr) {
            cout << current->data;
            if (current->prev != nullptr) {
                cout << " <-> ";
            }
            current = current->prev;
        }
        cout << endl;
    }

    // Destructor untuk membersihkan memori
    ~DoublyLinkedList() {
        Node* current = head;
        while (current != nullptr) {
            Node* nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};
int main() {
    DoublyLinkedList dll;

    // Input dari pengguna
    int elemen;
    cout << "Masukkan 4 elemen secara berurutan: " << endl;
    for (int i = 0; i < 4; i++) {
        cout << "Elemen " << (i + 1) << ": ";
        cin >> elemen;
        dll.insertLast_2311104008(elemen);
    }
    // Menampilkan elemen dari depan ke belakang
    dll.displayForward_2311104008();

    // Menampilkan elemen dari belakang ke depan
    dll.displayBackward_2311104008();

    return 0;
}
```

Output

```
Masukkan 4 elemen secara berurutan:
Elemen 1: 1
Elemen 2: 2
Elemen 3: 3
Elemen 4: 4
Daftar elemen dari depan ke belakang: 1 <-> 2 <-> 3 <-> 4
Daftar elemen dari belakang ke depan: 4 <-> 3 <-> 2 <-> 1
```

Kodingan ini mengimplementasikan Doubly Linked List dengan struktur Node yang menyimpan data integer dan dua pointer (prev dan next) untuk traversal dua arah. Kelas DoublyLinkedList mengelola list dengan pointer head dan tail, diinisialisasi dengan nullptr untuk menandakan list kosong. Fungsi **insertLast_2311104008** menambahkan elemen baru di akhir list,sementara **displayForward_2311104008** dan **displayBackward_2311104008** me nampilkan elemen dari depan ke belakang dan belakang ke depan, masing-masing. Destruktor membersihkan semua node untuk mencegah kebocoran memori. Dalam fungsi main, pengguna diminta memasukkan 4 elemen yang ditambahkan ke linked list, lalu ditampilkan dengan kedua fungsi display.