Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHub yang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. Asisten Praktikum : 4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folder dalam repository akan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwal praktikum adalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktu ini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem-solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

8. **Struktur Laporan Praktikum**

1. Cover :

LAPORAN PRAKTIKUM
Modul 6
DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:
KAFKA PUTRA RIYADI - 2311104041
Kelas:
SE 07-02

Dosen :
Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

2. Tujuan

1. Memahami konsep modul *linked list*.
2. Mengaplikasikan konsep *double linked list* dengan menggunakan *pointer* dan dengan bahasa C

3. Landasan Teori

Double Linked List adalah elemen-elemen yang dihubungkan dengan dua pointer dalam satu elemen dan list dapat melintas baik di depan atau belakang. Elemen double

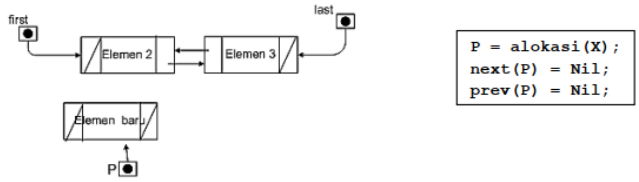
linked list terdiri dari empat bagian :

1. Bagian data informasi
2. Pointer next yang menunjuk ke elemen berikutnya
3. Pointer prev yang menunjuk ke elemen sebelumnya
4. Pointer *Last* : *pointer* pada *list* yang menunjuk pada elemen terakhir *list*

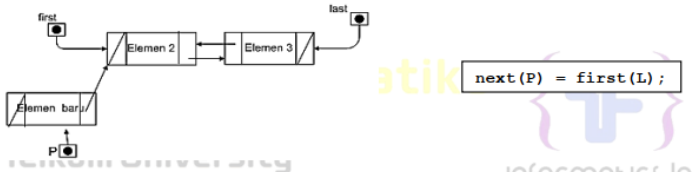
3.1.1 Insert

A. insert first

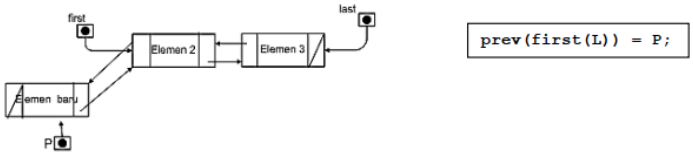
adalah operasi untuk menambahkan node baru di awal daftar, sehingga node baru ini menjadi elemen pertama (atau head) dari daftar tersebut. Setelah operasi ini, node yang sebelumnya menjadi head akan menjadi elemen kedua dalam daftar. Berikut adalah step atau langkah" dalam proses insert first :



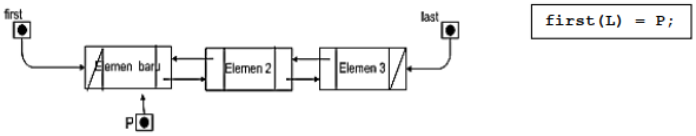
Gambar 6-3 *Double Linked list Insert First* 1



Gambar 6-4 *Double Linked list Insert First* 2



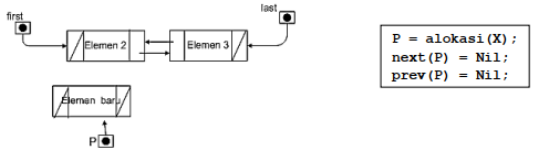
Gambar 6-5 *Double Linked list Insert First* 3



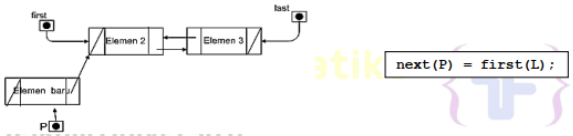
B. Insert last

adalah operasi untuk menambahkan node baru di awal (*head*) dari double linked list. Dalam double linked list, operasi ini membuat node baru menjadi elemen pertama dalam daftar, sementara node yang sebelumnya menjadi head akan bergeser ke posisi berikutnya.

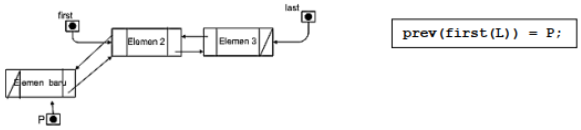
Berikut adalah sebuah gambaran langkah" dalam proses insert last:



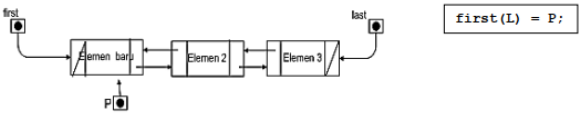
Gambar 6-3 *Double Linked list Insert First* 1



Gambar 6-4 *Double Linked list Insert First* 2



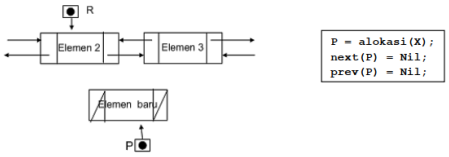
Gambar 6-5 *Double Linked list Insert First* 3



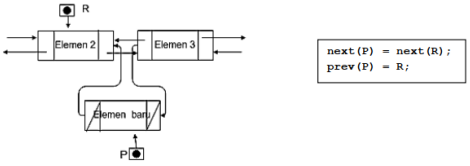
Gambar 6-6 *Double Linked list Insert First* 4

C. Insert after

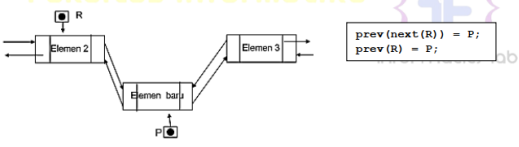
"Insert after" berarti menambahkan sesuatu setelah item tertentu dalam sebuah daftar atau dokumen. Misalnya, jika Anda memiliki daftar poin dan ingin menambahkan poin baru setelah poin yang sudah ada, Anda akan "insert after" poin tersebut. Dalam konteks pemrograman atau pengolahan data, ini bisa berarti menambahkan elemen baru ke dalam struktur data setelah elemen tertentu.
Berikut adalah sebuah gambaran langkah" dalam proses insert after:



Gambar 6-11 *Double Linked list Insert After* 1



Gambar 6-12 *Double Linked list Insert After* 2



Gambar 6-13 *Double Linked list Insert After* 3

D. Insert before

"Insert before" berarti menambahkan sesuatu sebelum item tertentu dalam sebuah daftar atau dokumen. Misalnya, jika Anda memiliki urutan poin dan ingin menambahkan poin baru sebelum poin yang sudah ada, Anda akan "insert before" poin tersebut.

Dalam konteks pemrograman atau pengolahan data, ini sering digunakan saat mengelola struktur data seperti daftar, array, atau linked list. Proses ini dapat melibatkan penempatan elemen baru di lokasi tertentu berdasarkan indeks atau posisi.

3.1.2 Delete

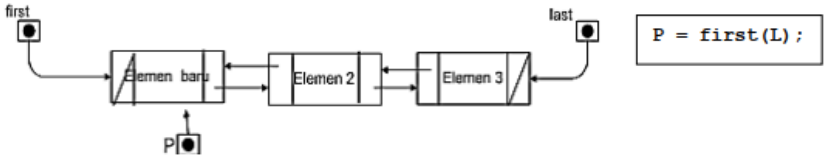
A. Delete first

"Delete first" adalah operasi pada struktur data linked list yang menghapus node pertama dari daftar.

Berikut adalah langkah-langkah dasar untuk operasi **delete first** pada linked list:

1. Identifikasi node pertama: Node pertama adalah yang terhubung langsung ke pointer atau referensi "head" dari linked list.
2. Perbarui pointer head: Pointer atau referensi "head" dari linked list diperbarui sehingga sekarang menunjuk ke node kedua dalam daftar.
3. Hapus node pertama: Setelah pointer atau referensi "head" diperbarui, node pertama sebelumnya tidak lagi memiliki koneksi dan bisa dihapus dari memori (di dalam bahasa pemrograman tertentu, ini dilakukan secara otomatis oleh garbage collector atau bisa dilakukan dengan eksplisit).

Dan ini adalah contoh proses atau Langkah" dari delete first



Gambar 6-14 *Double Linked list Delete First* 1

B. Delete last

"Delete last" adalah operasi dalam linked list yang menghapus node terakhir dari daftar. Operasi ini berguna untuk menghilangkan elemen paling akhir pada linked list tanpa mengubah node lainnya.

Berikut adalah langkah-langkah dasar untuk melakukan operasi **delete last**:

1. Temukan node sebelum node terakhir: Mulai dari node awal (head), jelajahi linked list hingga mencapai node yang berada tepat sebelum node terakhir (yaitu, node yang memiliki pointer next yang menunjuk ke node terakhir).
2. Hapus referensi ke node terakhir: Ubah pointer next dari node sebelum node terakhir sehingga menunjuk ke null atau kosong, menjadikannya node terakhir yang baru.
3. Hapus node terakhir lama: Setelah pointer diperbarui, node terakhir lama tidak

lagi terhubung dalam linked list dan bisa dihapus dari memori.

C. Delete after

"Delete after" adalah operasi pada linked list yang menghapus node yang berada setelah node tertentu. Dalam operasi ini, kita menghapus node berikutnya dari node yang ditentukan tanpa memengaruhi node lainnya dalam linked list.

Berikut langkah-langkah dasar untuk melakukan operasi **delete after**:

1. Temukan node target: Pilih node yang berada tepat sebelum node yang ingin dihapus.
2. Identifikasi node yang akan dihapus: Dari node target, cari node berikutnya (node yang akan dihapus).
3. Perbarui pointer: Ubah pointer dari node target agar melewati node yang akan dihapus dan langsung menunjuk ke node setelahnya. Dengan cara ini, node yang dihapus akan dilewatkan dalam urutan linked list.
4. Hapus node berikutnya: Setelah pointer diperbarui, node yang dihapus tidak lagi terhubung dalam linked list dan bisa dilepaskan dari memori (dalam bahasa pemrograman tertentu, ini dilakukan otomatis atau secara eksplisit dihapus).

D. Delete before

"Delete before" adalah operasi dalam linked list yang menghapus node yang berada tepat sebelum node tertentu. Operasi ini sedikit lebih rumit dibandingkan operasi delete lainnya karena membutuhkan akses ke node sebelum node target.

Berikut adalah langkah-langkah dasar untuk operasi **delete before**:

1. Temukan node sebelum node yang akan dihapus: Mulai dari node awal (head), jelajahi linked list hingga menemukan node yang berada dua langkah sebelum node target.
2. Identifikasi node yang akan dihapus: Dari node tersebut, pilih node berikutnya sebagai node yang akan dihapus (yaitu, node tepat sebelum node target).
3. Perbarui pointer: Ubah pointer dari node sebelumnya agar melewati node yang akan dihapus dan langsung menunjuk ke node target. Dengan cara ini, node yang dihapus akan dilewati dalam urutan linked list.
4. Hapus node sebelum target: Setelah pointer diperbarui, node yang dihapus tidak lagi terhubung dalam linked list dan bisa dilepaskan dari memori.

E. Update, View, dan Searching

Update: Mengubah nilai dari node tertentu.

- Langkah: Telusuri hingga ke node yang ingin diubah, lalu ganti nilainya.

View: Menampilkan semua elemen dalam linked list.

- Langkah: Telusuri dari awal hingga akhir dan cetak nilai di setiap node.

Searching: Mencari node berdasarkan nilai tertentu.

- Langkah: Telusuri setiap node dan periksa nilainya sampai nilai yang dicari ditemukan.

4. **Guided**

Codingannya saya bagi menjadi 2 gambar agar tidak ngeblur dan tidak terlalu makan tempat

```cpp
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    // Constructor untuk inisialisasi head dan tail
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menambahkan elemen di depan list
    void insert(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode; // Jika list kosong, tail juga mengarah ke node baru
        }
        head = newNode;
    }

    // Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }
```

Telkom University Purwokerto

Lanjutan dari codingan sebelumnya.

```cpp
    // Fungsi untuk menampilkan semua elemen di list
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
                list.insert(data);
                break;
            }
            case 2: {
                list.deleteNode();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                bool updated = list.update(oldData, newData);
                if (!updated) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
    return 0;
}
```

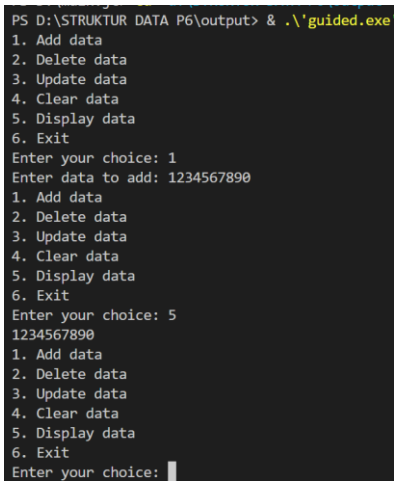
Code program C++ diatas mengimplementasikan Doubly Linked List (DLL) dan menyediakan menu interaktif untuk pengguna untuk melakukan berbagai operasi pada linked list tersebut. Berikut adalah penjelasan bagian-bagian utama dari kode tersebut:

1. Kelas Node
   - Definisi: Mewakili elemen dari doubly linked list.
   - Atribut:
     - data: Menyimpan nilai yang disimpan dalam node.
     - prev: Pointer ke node sebelumnya.
     - next: Pointer ke node berikutnya.

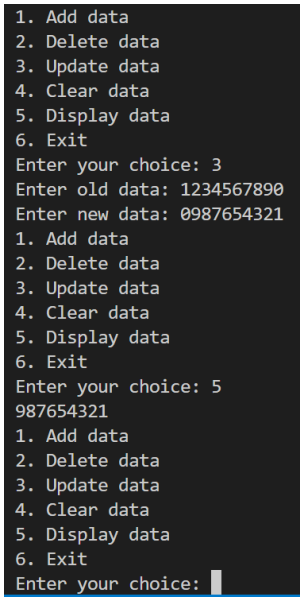
2. Kelas DoublyLinkedList
   - Atribut:
     - head: Pointer ke node pertama dalam list.
     - tail: Pointer ke node terakhir dalam list.
   - Metode:
     - Konstruktor: Menginisialisasi head dan tail ke nullptr, menandakan list kosong.
     - insert(int data): Menambahkan node baru di depan list. Jika list kosong, node baru menjadi tail. Jika tidak, head yang lama diupdate untuk menunjuk ke node baru.
     - deleteNode(): Menghapus node pertama dari list. Jika list kosong, tidak ada yang dihapus. Jika hanya ada satu node, tail juga diupdate ke nullptr.
     - update(int oldData, int newData): Mengupdate data pada node yang ada. Jika node dengan oldData ditemukan, nilainya diubah menjadi newData dan mengembalikan true; jika tidak, mengembalikan false.
     - deleteAll(): Menghapus semua node dari list. Mengiterasi melalui semua node dan menghapusnya satu per satu, kemudian mengatur head dan tail ke nullptr.
     - display(): Menampilkan semua elemen dalam list. Mengiterasi dari head hingga tail dan mencetak nilai data setiap node.

3. Fungsi main()
   - Menu Interaktif: Program berfungsi dalam loop tak terbatas untuk memberikan menu pilihan kepada pengguna:
     1. Add data: Menggunakan insert() untuk menambahkan data ke list.
     2. Delete data: Menggunakan deleteNode() untuk menghapus node pertama.
     3. Update data: Menggunakan update() untuk mengganti data yang sudah ada.
     4. Clear data: Menggunakan deleteAll() untuk menghapus semua node.
     5. Display data: Menggunakan display() untuk menampilkan semua data dalam list.
     6. Exit: Menghentikan program.

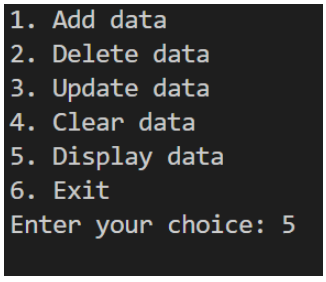
Outputan codingan diatas apabila kita menambah data atau Add data dan langsung display data atau menampilkan data yang telah kita masukan sebelumnya :



Outputan setelah meng-update data dari data sebelumnya:



Fungsi dari delete data dan clear data yaitu sama untuk menghilangkan atau menghapus data yang telah kita masukkan sebelumnya, dan ini hasil outputannya:



fungsi exit nya yaitu untuk keluar dari program

Telkom University Purwokerto

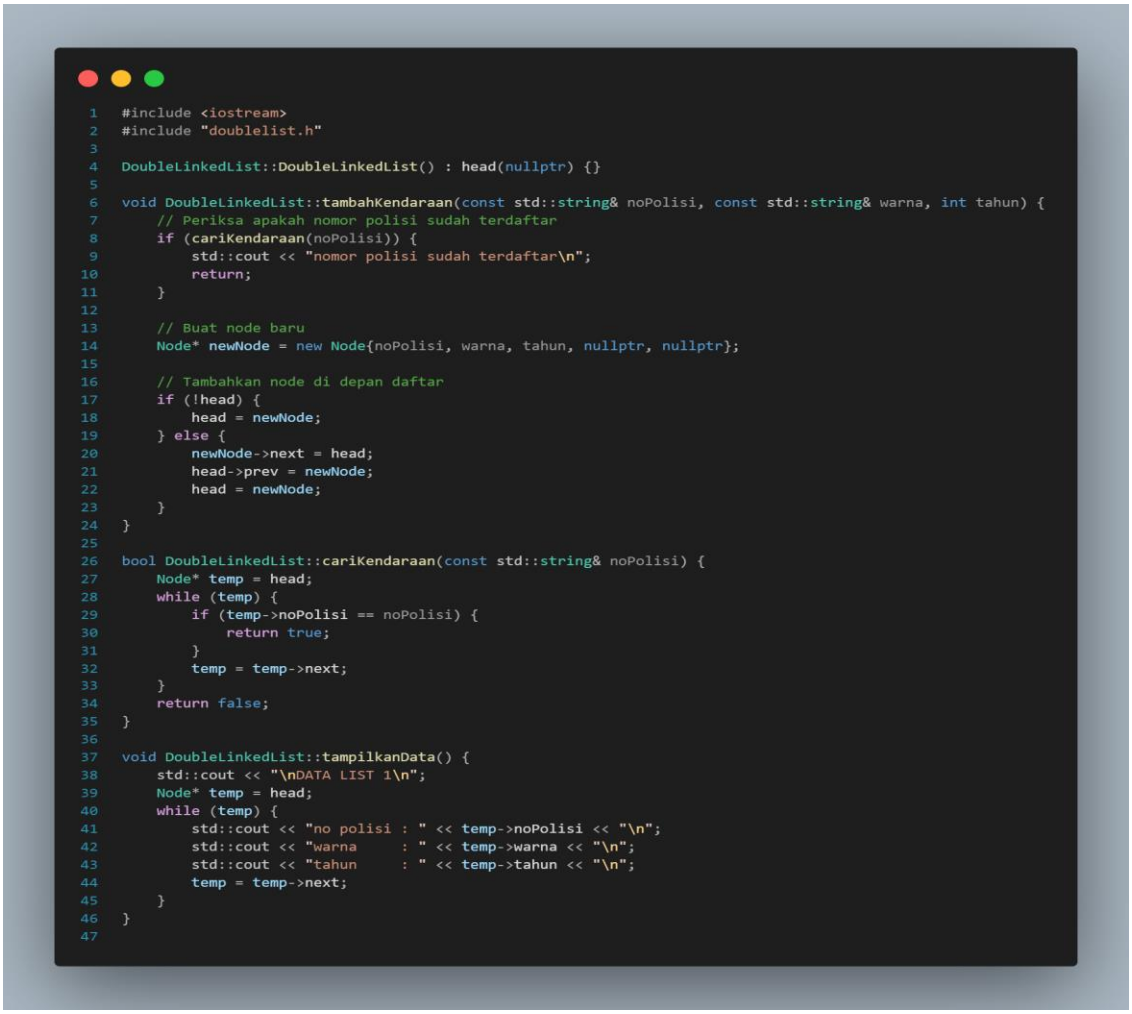
5. Unguided

NO.1.

Doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>

struct Node {
    std::string noPolisi;
    std::string warna;
    int tahun;
    Node* next;
    Node* prev;
};

class DoubleLinkedList {
public:
    DoubleLinkedList();
    void tambahKendaraan(const std::string& noPolisi, const std::string& warna, int tahun);
    bool cariKendaraan(const std::string& noPolisi);
    void tampilkanData();

private:
    Node* head;
};

#endif // DOUBLELIST_H
```

Doublelist.cpp



```
#include <iostream>
#include "doublelist.h"

DoubleLinkedList::DoubleLinkedList() : head(nullptr) {}

void DoubleLinkedList::tambahKendaraan(const std::string& noPolisi, const std::string& warna, int tahun) {
    // Periksa apakah nomor polisi sudah terdaftar
    if (cariKendaraan(noPolisi)) {
        std::cout << "nomor polisi sudah terdaftar\n";
        return;
    }

    // Buat node baru
    Node* newNode = new Node{noPolisi, warna, tahun, nullptr, nullptr};

    // Tambahkan node di depan daftar
    if (!head) {
        head = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

bool DoubleLinkedList::cariKendaraan(const std::string& noPolisi) {
    Node* temp = head;
    while (temp) {
        if (temp->noPolisi == noPolisi) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

void DoubleLinkedList::tampilkanData() {
    std::cout << "\nDATA LIST 1\n";
    Node* temp = head;
    while (temp) {
        std::cout << "no polisi : " << temp->noPolisi << "\n";
        std::cout << "warna     : " << temp->warna << "\n";
        std::cout << "tahun     : " << temp->tahun << "\n";
        temp = temp->next;
    }
}
```

Telkom University Purwokerto

Main.cpp

```cpp
#include <iostream>
#include "doublelist.h"
#include "doublelist.cpp"


int main() {
    DoubleLinkedList list;
    std::string noPolisi, warna;
    int tahun;

    // Input pertama
    std::cout << "masukkan nomor polisi: ";
    std::cin >> noPolisi;
    std::cout << "masukkan warna kendaraan: ";
    std::cin >> warna;
    std::cout << "masukkan tahun kendaraan: ";
    std::cin >> tahun;
    list.tambahKendaraan(noPolisi, warna, tahun);

    // Input kedua
    std::cout << "\nmasukkan nomor polisi: ";
    std::cin >> noPolisi;
    std::cout << "masukkan warna kendaraan: ";
    std::cin >> warna;
    std::cout << "masukkan tahun kendaraan: ";
    std::cin >> tahun;
    list.tambahKendaraan(noPolisi, warna, tahun);

    // Input ketiga (cek duplikat)
    std::cout << "\nmasukkan nomor polisi: ";
    std::cin >> noPolisi;
    std::cout << "masukkan warna kendaraan: ";
    std::cin >> warna;
    std::cout << "masukkan tahun kendaraan: ";
    std::cin >> tahun;
    list.tambahKendaraan(noPolisi, warna, tahun);

    // Input keempat
    std::cout << "\nmasukkan nomor polisi: ";
    std::cin >> noPolisi;
    std::cout << "masukkan warna kendaraan: ";
    std::cin >> warna;
    std::cout << "masukkan tahun kendaraan: ";
    std::cin >> tahun;
    list.tambahKendaraan(noPolisi, warna, tahun);

    // Tampilkan data
    list.tampilkanData();

    return 0;
}
```

Dan ini hasil outputannya:

```
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1
no polisi : D004
warna     : kuning
tahun     : 90
no polisi : D003
warna     : putih
tahun     : 70
no polisi : D001
warna     : hitam
tahun     : 90
```

NO.2

```cpp
#include <iostream>
#include <string>
using namespace std;

// Struktur data untuk informasi kendaraan
struct infotype {
    string NomorPolisi;
    string Warna;
    int Tahun;
};

// Struktur Node
struct Node {
    infotype info;
    Node* next;
};

// Kelas List yang menyimpan head pointer
class List {
private:
    Node* first;

public:
    List() : first(nullptr) {} // Konstruktor untuk inisialisasi list kosong

    // Fungsi untuk menambahkan data ke list
    void addData(const string& nomorPolisi, const string& warna, int tahun) {
        Node* newNode = new Node;
        newNode->info.NomorPolisi = nomorPolisi;
        newNode->info.Warna = warna;
        newNode->info.Tahun = tahun;
        newNode->next = first;
        first = newNode;
    }

    // Fungsi untuk mencari data berdasarkan NomorPolisi
    Node* findElm(const string& nomorPolisiDicari) {
        Node* P = first;
        while (P != nullptr) {
            if (P->info.NomorPolisi == nomorPolisiDicari) {
                return P; // Mengembalikan alamat node jika ditemukan
            }
            P = P->next;
        }
        return nullptr; // Mengembalikan nullptr jika tidak ditemukan
    }
};

int main() {
    List L;

    // Menambahkan beberapa data ke dalam list
    L.addData("D001", "hitam", 90);
    L.addData("D002", "merah", 91);
    L.addData("D003", "biru", 92);

    // Input dari pengguna untuk mencari nomor polisi
    string nomorPolisiDicari;
    cout << "Masukkan Nomor Polisi yang dicari: ";
    cin >> nomorPolisiDicari;

    // Mencari data di list
    Node* hasil = L.findElm(nomorPolisiDicari);
    if (hasil != nullptr) {
        cout << "Nomor Polisi: " << hasil->info.NomorPolisi << endl;
        cout << "Warna       : " << hasil->info.Warna << endl;
        cout << "Tahun       : " << hasil->info.Tahun << endl;
    } else {
        cout << "Nomor Polisi tidak ditemukan." << endl;
    }

    return 0;
}
```

Dan ini hasil outputannya:

```
Masukkan Nomor Polisi yang dicari: D001
Nomor Polisi: D001
Warna       : hitam
Tahun       : 90
PS D:\STRUKTUR DATA P6\output>
```

NO.3

Pada jawaban soal no.3 foto codingannya saya bagi 2 agar tidak blur/terlalu kecil jika dijadikan 1

```cpp
#include <iostream>
#include <string>
using namespace std;

// Struktur node untuk linked list
struct Node {
    string nomor_polisi;
    string warna;
    string tahun;
    Node* next;
};

// Fungsi untuk membuat node baru
Node* createNode(const string& nomor, const string& warna, const string& tahun) {
    Node* newNode = new Node;
    newNode->nomor_polisi = nomor;
    newNode->warna = warna;
    newNode->tahun = tahun;
    newNode->next = nullptr;
    return newNode;
}

// Prosedur untuk menampilkan daftar
void displayList(Node* head) {
    Node* current = head;
    cout << "DATA LIST 1\n \n";
    while (current != nullptr) {
        cout << "Nomor Polisi : " << current->nomor_polisi << endl;
        cout << "Warna        : " << current->warna << endl;
        cout << "Tahun        : " << current->tahun << endl;
        current = current->next;
    }
}


// Prosedur untuk menghapus node pertama
void deleteFirst(Node*& head) {
    if (head == nullptr) return;
    Node* temp = head;
    head = head->next;
    delete temp;
}

// Prosedur untuk menghapus node terakhir
void deleteLast(Node*& head) {
    if (head == nullptr) return;
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        return;
    }
    Node* current = head;
    while (current->next->next != nullptr) {
        current = current->next;
    }
    delete current->next;
    current->next = nullptr;
}

// Prosedur untuk menghapus node setelah node tertentu
void deleteAfter(Node* prec) {
    if (prec == nullptr || prec->next == nullptr) return;
    Node* temp = prec->next;
    prec->next = temp->next;
    delete temp;
```

Lanjutan codingan dari gambar diatas

```cpp
Node* findNode(Node* head, const string& nomor_polisi) {
    Node* current = head;
    while (current != nullptr) {
        if (current->nomor_polisi == nomor_polisi) {
            return current;
        }
        current = current->next;
    }
    return nullptr;
}

// Fungsi untuk menghapus node berdasarkan nomor polisi
void deleteByLicenseNumber(Node*& head, const string& nomor_polisi) {
    if (head == nullptr) return;

    // Jika elemen yang akan dihapus adalah elemen pertama
    if (head->nomor_polisi == nomor_polisi) {
        deleteFirst(head);
        cout << "Data dengan nomor polisi " << nomor_polisi << " berhasil dihapus.\n \n";
        return;
    }

    // Mencari node sebelum node yang akan dihapus
    Node* current = head;
    while (current->next != nullptr && current->next->nomor_polisi != nomor_polisi) {
        current = current->next;
    }

    // Jika elemen ditemukan
    if (current->next != nullptr) {
        Node* temp = current->next;
        current->next = temp->next;
        delete temp;
        cout << "Data dengan nomor polisi " << nomor_polisi << " berhasil dihapus.\n \n";
    } else {
        cout << "Nomor Polisi " << nomor_polisi << " tidak ditemukan.\n";
    }
}

// Fungsi utama
int main() {
    // Membuat list
    Node* head = createNode("D004", "kuning", "90");
    head->next = createNode("D003", "merah", "89");
    head->next->next = createNode("D001", "hitam", "90");

    // Menampilkan list awal
    cout << "";
    displayList(head);

    // Meminta nomor polisi yang akan dihapus
    string nomor_dihapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> nomor_dihapus;

    // Menghapus elemen berdasarkan nomor polisi
    deleteByLicenseNumber(head, nomor_dihapus);

    // Menampilkan list setelah penghapusan
    cout << "";
    displayList(head);

    return 0;
}

```

Dan ini hasil outputannya:

```
Masukkan Nomor Polisi yang akan dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

Nomor Polisi : D004
Warna        : kuning
Tahun        : 90
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90
PS D:\STRUKTUR DATA P6\output>
```

6. Kesimpulan

Jadi Kesimpulan nya yaittu Double Linked List adalah struktur data yang terdiri dari elemen-elemen yang saling terhubung dengan dua pointer: satu menunjuk ke elemen berikutnya (next) dan satu lagi ke elemen sebelumnya (prev). Ini memungkinkan navigasi baik maju maupun mundur di dalam daftar. Setiap elemen pada double linked list terdiri dari data, pointer next, pointer prev, dan pointer last untuk menunjuk elemen terakhir.

Operasi dasar dalam double linked list mencakup:

1. **Insert (Menambahkan Elemen):**
    - *Insert First*: Menambahkan elemen di awal daftar.
    - *Insert Last*: Menambahkan elemen di akhir daftar.
    - *Insert After*: Menambahkan elemen setelah elemen tertentu.
    - *Insert Before*: Menambahkan elemen sebelum elemen tertentu.
2. **Delete (Menghapus Elemen):**
    - *Delete First*: Menghapus elemen pertama.
    - *Delete Last*: Menghapus elemen terakhir.
    - *Delete After*: Menghapus elemen setelah elemen tertentu.
    - *Delete Before*: Menghapus elemen sebelum elemen tertentu.
3. **Operasi Lainnya:**
    - *Update*: Mengubah nilai pada elemen tertentu.
    - *View*: Menampilkan semua elemen dalam daftar.
    - *Searching*: Mencari elemen berdasarkan nilai tertentu.