LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLY LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :
Nama: Dina Nadhyfa
NIM : 103112430052

Dosen
Fahrudin Mukti Wibowo

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Doubly linked list adalah jenis struktur data linked list yang terdiri dari serangkaian node, di mana setiap node memiliki tiga bagian utama: data, pointer ke node berikutnya (next), dan pointer ke node sebelumnya (prev). Dengan adanya dua pointer ini, doubly linked list memungkinkan traversing data secara dua arah, baik maju maupun mundur, sehingga lebih fleksibel dibandingkan single linked list yang hanya bisa dilintasi satu arah. Struktur ini biasanya memiliki pointer khusus yaitu head, yang menunjuk ke node pertama, dan tail, yang menunjuk ke node terakhir dalam list.

Operasi dasar dalam doubly linked list meliputi penyisipan dan penghapusan node di berbagai posisi, seperti di awal (insert first), akhir (insert last), dan posisi tertentu di tengah list. Operasi penghapusan juga bisa dilakukan di posisi awal, akhir, maupun node tertentu berdasarkan data. Keunggulan utama doubly linked list adalah kemudahan navigasi dua arah yang mempermudah proses pencarian dan manipulasi data, meskipun penggunaan memori relatif lebih besar dibandingkan single linked list karena adanya dua pointer pada setiap node.

Implementasi doubly linked list banyak diaplikasikan dalam berbagai bidang pemrograman, terutama ketika fitur navigasi dua arah diperlukan, seperti pada undo-redo dalam editor teks, browsing history, atau manajemen buffer di sistem operasi. Struktur ini merupakan dasar penting dalam pengembangan algoritma dan aplikasi yang memerlukan struktur data yang dinamis dan efisien.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

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

struct Node
{
    int data;
    Node* prev;
    Node* next;
};

Node*ptr_first = NULL;
Node*ptr_last = NULL;

void add_first(int value)
{
    Node *newNode = new Node{value, NULL, ptr_first};

    if (ptr_first == NULL)
    {
        ptr_last = newNode;
    }
```

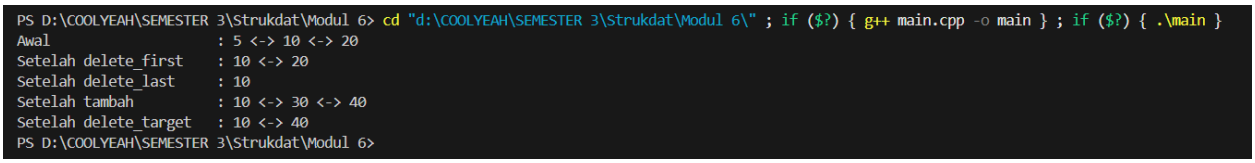
```
    else
    {
        ptr_first->prev = newNode;
    }
    ptr_first = newNode;
}

void add_last(int value)
{
    Node *newNode = new Node{value, ptr_last, NULL};

    if (ptr_last == NULL)
    {
        ptr_first = newNode;
    }
    else
    {
        ptr_last->next = newNode;
    }
    ptr_last = newNode;
}

void add_target(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        if (current == ptr_last)
        {
            add_last(newValue);
        }
        else
        {
            Node *newNode = new Node{newValue, current, current->next};
            current->next->prev = newNode;
            current->next = newNode;
        }
    }
}

void view()
{
    Node *current = ptr_first;
```

```
    if (current == NULL)
    {
        cout << "List Kosong\n";
        return;
    }
    while (current != NULL)
    {
        cout << current->data << (current->next != NULL ? " <-> " : "");
        current = current->next;
    }
    cout << endl;

}

void delete_first()
{
    if (ptr_first == NULL)
    return;

    Node *temp = ptr_first;
    if (ptr_first == ptr_last)
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_first = ptr_first->next;
        ptr_first->prev = NULL;
    }
    delete temp;
}

void delete_last()
{
    if (ptr_last == NULL)
    return;

    Node *temp = ptr_last;

    if (ptr_first == ptr_last)
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_last = ptr_last->prev;
```

```
        ptr_last->next = NULL;
    }
    delete temp;
}

void delete_target(int targetValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        if (current == ptr_first)
        {
            delete_first();
            return;
        }
        if (current == ptr_last)
        {
            delete_last();
            return;
        }

        current->prev->next = current->next;
        current->next->prev = current->prev;
        delete current;
    }
}

void edit_node(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        current->data = newValue;
    }
}

int main()
{
```

```
    add_first(10);
    add_first(5);
    add_last(20);
    cout << "Awal\t\t\t: ";
    view();

    delete_first();
    cout << "Setelah delete_first\t: ";
    view();
    delete_last();
    cout << "Setelah delete_last\t: ";
    view();

    add_last(30);
    add_last(40);
    cout << "Setelah tambah\t\t: ";
    view();

    delete_target(30);
    cout << "Setelah delete_target\t: ";
    view();
}
```

Screenshots Output



Deskripsi:

Program ini merupakan implementasi Doubly Linked List sederhana untuk mengelola data integer dengan operasi-operasi dasar. Program menggunakan struktur Node yang memiliki tiga komponen: data integer, pointer prev ke node sebelumnya, dan pointer next ke node berikutnya.

Program menyediakan berbagai fungsi manipulasi data seperti add_first dan add_last untuk menambah node di awal dan akhir list, add_target untuk menyisipkan node setelah nilai tertentu, serta delete_first, delete_last, dan delete_target untuk menghapus node dari berbagai posisi. Terdapat juga fungsi view untuk menampilkan seluruh data dalam format linked list dengan simbol "↔" sebagai pemisah, dan edit_node untuk mengubah nilai node tertentu.

Dalam fungsi main, program mendemonstrasikan penggunaan berbagai operasi tersebut dengan menambah data 5, 10, 20, kemudian menghapus node pertama dan terakhir, menambah data 30 dan 40, serta menghapus node dengan nilai 30. Setiap langkah operasi ditampilkan untuk menunjukkan perubahan yang terjadi pada struktur data linked list.

Program ini cocok untuk memahami konsep dasar operasi pada doubly linked list dengan visualisasi yang jelas.

D. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

Doublylist.h

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <iostream>
using namespace std;

struct infotype {
    string nopol;
    string warna;
    int thnBuat;
};

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
    address prev;
};

struct List {
    address First;
    address Last;
};

// Prosedur dan fungsi yang harus diimplementasi
void CreateList(List &L);
address alokasi(infotype X);
void dealokasi(address &P);
void printInfo(List L);
void insertLast(List &L, address P);
address findElm(List L, string nopol);  // Fungsi baru sesuai seperti
yang ada di soal nomor 2

// Fungsi delete baru soal nomor 3
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
bool deleteByNopol(List &L, string nopol);
```

```
// Fungsi tambahan untuk membantu pengecekan nopol
bool isNopolExist(List L, string nopol);

#endif
```

Doublylist.cpp

```
#include "Doublylist.h"

// Membuat list kosong
void CreateList(List &L) {
    L.First = NULL;
    L.Last = NULL;
}

// Mengalokasikan memori untuk elemen baru
address alokasi(infotype X) {
    address P = new ElmList;
    P->info = X;
    P->next = NULL;
    P->prev = NULL;
    return P;
}

// Dealokasi memori
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Menampilkan semua informasi dalam list
void printInfo(List L) {
    if (L.First == NULL) {
        cout << "List kosong" << endl;
        return;
    }

    address P = L.First;
    while (P != NULL) {
        cout << "nopol    : " << P->info.nopol << endl;
        cout << "warna    : " << P->info.warna << endl;
        cout << "tahun    : " << P->info.thnBuat << endl;
        cout << endl;
        P = P->next;
    }
}
```

```
// Menyisipkan elemen di akhir list
void insertLast(List &L, address P) {
    if (L.First == NULL) {
        // List kosong
        L.First = P;
        L.Last = P;
    } else {
        // List tidak kosong
        L.Last->next = P;
        P->prev = L.Last;
        L.Last = P;
    }
}

// Fungsi baru soal nomor 2: Mencari elemen berdasarkan nomor polisi
address findElm(List L, string nopol) {
    address P = L.First;
    while (P != NULL) {
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return NULL; // Tidak ditemukan
}

// Fungsi tambahan soal nomor 2: Mengecek apakah nomor polisi sudah ada
bool isNopolExist(List L, string nopol) {
    return findElm(L, nopol) != NULL;
}

// fungsi baru soal nomor 3: Menghapus elemen pertama
void deleteFirst(List &L, address &P) {
    if (L.First == NULL) {
        P = NULL;
        return;
    }

    P = L.First;
    if (L.First == L.Last) { // Hanya satu elemen
        L.First = NULL;
        L.Last = NULL;
    } else { // Lebih dari satu elemen
        L.First = L.First->next;
        L.First->prev = NULL;
        P->next = NULL;
    }
```

```
}

// fungsi baru soal nomor 3: Menghapus elemen terakhir
void deleteLast(List &L, address &P) {
    if (L.Last == NULL) {
        P = NULL;
        return;
    }

    P = L.Last;
    if (L.First == L.Last) { // Hanya satu elemen
        L.First = NULL;
        L.Last = NULL;
    } else { // Lebih dari satu elemen
        L.Last = L.Last->prev;
        L.Last->next = NULL;
        P->prev = NULL;
    }
}

//fungsi baru soal nomor 3: Menghapus elemen setelah Prec
void deleteAfter(address Prec, address &P) {
    if (Prec == NULL || Prec->next == NULL) {
        P = NULL;
        return;
    }

    P = Prec->next;
    Prec->next = P->next;

    if (P->next != NULL) {
        P->next->prev = Prec;
    }

    P->next = NULL;
    P->prev = NULL;
}

// Fungsi utama untuk menghapus berdasarkan nomor polisi
bool deleteByNopol(List &L, string nopol) {
    address P = findElm(L, nopol);

    if (P == NULL) {
        return false; // Data tidak ditemukan
    }

    address deletedNode;
```

```
    if (P == L.First) {
        // Hapus elemen pertama
        deleteFirst(L, deletedNode);
        dealokasi(deletedNode);
    } else if (P == L.Last) {
        // Hapus elemen terakhir
        deleteLast(L, deletedNode);
        dealokasi(deletedNode);
    } else {
        // Hapus elemen di tengah
        deleteAfter(P->prev, deletedNode);
        dealokasi(deletedNode);
    }

    return true;
}
```

Main.cpp

```
#include <iostream>
#include "Doublylist.h"
using namespace std;

int main() {
    List L;
    CreateList(L);

    int jumlahData = 0;
    const int MAX_DATA = 3; // Sesuai contoh output

    // Input data kendaraan
    while (jumlahData < MAX_DATA) {
        infotype kendaraan;

        cout << "masukkan nomor polisi: ";
        cin >> kendaraan.nopol;
        cout << "masukkan warna kendaraan: ";
        cin >> kendaraan.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> kendaraan.thnBuat;
        cout << endl;

        // Cek apakah nomor polisi sudah terdaftar
        if (isNopolExist(L, kendaraan.nopol)) {
            cout << "nomor polisi sudah terdaftar" << endl;
            cout << endl;
        } else {
```

```
            // Jika belum terdaftar, tambahkan ke list
            address P = alokasi(kendaraan);
            insertLast(L, P);
            jumlahData++;
        }
    }

    cout << "DATA LIST 1" << endl;
    cout << endl;
    printInfo(L);

    // Pencarian nomor polisi
    string nopolCari;
    cout << "Masukkan nomor polisi yang dicari: ";
    cin >> nopolCari;

    address hasilCari = findElm(L, nopolCari);

    if (hasilCari != NULL) {
        cout << "nopol    : " << hasilCari->info.nopol << endl;
        cout << "warna    : " << hasilCari->info.warna << endl;
        cout << "tahun    : " << hasilCari->info.thnBuat << endl;
    } else {
        cout << "Nomor polisi " << nopolCari << " tidak ditemukan" <<
endl;
    }

      // Menghapus data berdasarkan nomor polisi
    string nopolHapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> nopolHapus;

        address P = findElm(L, nopolHapus);

    if (P != NULL) {
        deleteByNopol(L, nopolHapus);
        cout << "Data dengan nomor polisi " << nopolHapus << " berhasil
dihapus." << endl;
    }

    cout << endl;

    cout << "DATA LIST 1" << endl;

    printInfo(L);

    return 0;
}
```

Screenshots Output

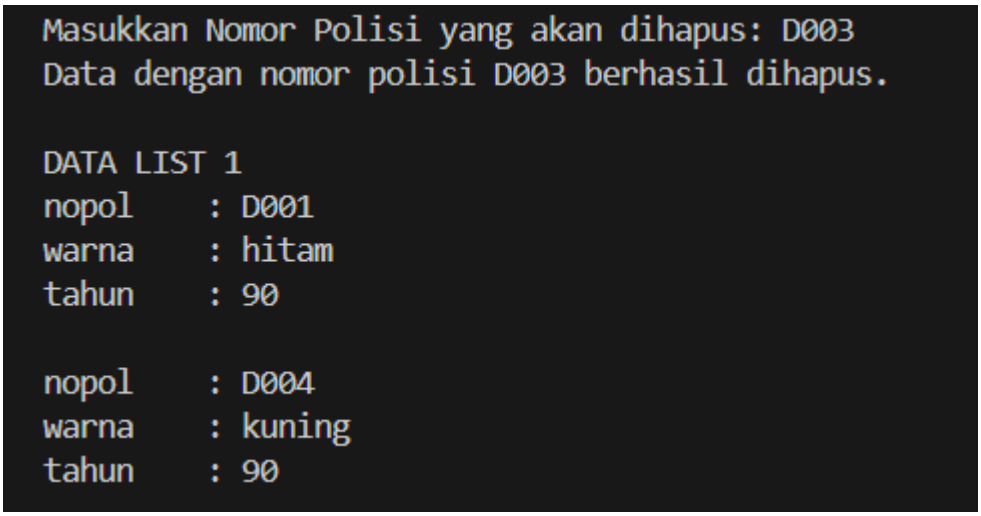
Nomor 1

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 6 Soal1> g++ main.cpp Doublylist.cpp -o main
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 6 Soal1> ./main
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

nopol    : D001
warna    : hitam
tahun    : 90

nopol    : D003
warna    : putih
tahun    : 70

nopol    : D004
warna    : kuning
tahun    : 90
```

Nomor 2

```
Masukkan nomor polisi yang dicari: D001
nopol    : D001
warna    : hitam
tahun    : 90
```

Nomor 3



Deskripsi:

Program di atas merupakan implementasi ADT Doubly Linked List untuk sistemn manajemen data kendaraan yang memungkinkan pengelolaan data kendaraan secara efisien. Program memiliki fitur utama untuk menambah, mencari, menghapus, dan menampilkan data kendaraan yang terdiri dari nomor polisis, warna, dan tahun pembuatan.

Struktur data yang digunakan adalah doubly linked list dimana setiap elemen memiliki pointer ke elemen sebelumnya dan berikutnya, memungkinkan traversal dua arah. Program dilengkapi dengan validasi untuk mencegah duplikasi nomor polisi dan manajemen memori yang aman dengan dealokasi setelah penghapusan.

Alur program dimulai dengan input 3 data kendaraan dengan oengecekan duplikasi, kemudian menampilkan semua data, melakukan pencarian berdasarkan nomor polisis, serta menghapus data tertentu dengan konfirmasi keberhasilan.

E. Kesimpulan

Berdasarkan seluruh diskusi yang telah dilakukan, dapat disimpulkan bahwa Anda telah mempelajari konsep mendalam tentang implementasi Abstract Data Type (ADT) Doubly Linked List beserta operasi-operasi fundamentalnya. Mulai dari pembuatan struktur data dasar dengan node yang memiliki pointer prev dan next, implementasi fungsi-fungsi essensial seperti penambahan data di awal (add_first), akhir (add_last), dan posisi tertentu (add_target), hingga operasi penghapusan data (delete_first, delete_last, delete_target) dan traversal data (view). Anda juga telah memahami penerapan praktis ADT dalam studi kasus manajemen data kendaraan yang meliputi validasi duplikasi nomor polisi, pencarian data, serta mekanisme penghapusan dengan penanganan memory management yang tepat. Seluruh pembahasan menunjukkan penguasaan konsep linked list dua arah yang mencakup aspek teoritis struktur data hingga implementasi nyata dalam pemecahan masalah programming.

F. Referensi

https://rumahcoding.co.id/linked-list-pengertian-dan-implementasi-dasar/

https://tita.lecturer.pens.ac.id/ASD/M6_M7_DLL_Insert/Double%20Linked%20List.pdf

https://www.slideshare.net/slideshow/makalah-double-linked-list-bahasa-cdocx/254436624

https://en.wikipedia.org/wiki/Doubly_linked_list