LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL IV
DOUBLY LINKED LIST

Disusun Oleh :
NAMA : Ridha Akifah
NIM : 103112400132

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Doubly Linked List adalah jenis *linked list* di mana setiap elemennya memiliki dua penunjuk (successor): satu penunjuk, disebut next, yang menunjuk ke elemen sesudahnya, dan satu penunjuk lainnya, disebut prev, yang menunjuk ke elemen sebelumnya. Struktur ini memungkinkan iterasi atau penelusuran list dari dua arah (maju dan mundur), yang menjadikannya lebih efisien dalam proses akses elemen dibandingkan *Singly Linked List*. List ini juga memiliki dua *successor* utama yang menunjuk ke elemen awal (first) dan elemen akhir (last). Dengan adanya penunjuk prev, operasi seperti Insert Before dan Delete Before menjadi lebih mudah diimplementasikan karena tidak memerlukan penelusuran dari awal list untuk menemukan elemen sebelumnya.

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();
```

```
    return 0;

}
```

Screenshoot Output

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS D:\C++> cd "d:\C++\Modul 6\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Awal                    : 5 <-> 10 <-> 20
Setelah delete_first    : 10 <-> 20
Setelah delete_last     : 10
Setelah tambah          : 10 <-> 30 <-> 40
Setelah delete_target   ; 10 <-> 40
PS D:\C++\Modul 6>
```

Deskripsi:

Program ini mengimplementasikan Doubly Linked List menggunakan pointer global. Program mendukung operasi dasar manajemen list: penambahan (add_first, add_last, add_target), penghapusan (delete_first, delete_last, delete_target), pengeditan (edit_node), dan penampilan (view) data list.

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

Unguided

Doublylist.h

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H
#include <iostream>
#include <string>
using namespace std;
struct kendaraan
{
    string nopol;
    string warna;
```

```
    int thnBuat;
};
typedef kendaraan infotype;
struct ElmList
{
    infotype info;
    ElmList *next;
    ElmList *prev;
};
typedef ElmList *address;
struct List
{
    address First;
    address Last;
};
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);
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
#endif
```

Doublylist.cpp

```cpp
#include "Doublylist.h"

void CreateList(List &L) { L.First = L.Last = NULL; }

address alokasi(infotype x)

{

    address P = new ElmList;

    P->info = x;

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

    return P;

}

void dealokasi(address &P)

{

    delete P;

    P = NULL;

}

void printInfo(List L)

{

    if (!L.First)

    {

        cout << "List kosong\n";

        return;

    }

    address P = L.First;

    int i = 1;

    cout << "\nDATA LIST:\n";

    while (P)

    {

        cout << "Data ke-" << i++ << endl;

        cout << "Nomor Polisi : " << P->info.nopol << endl;

        cout << "Warna : " << P->info.warna << endl;
```

```
        cout << "Tahun : " << P->info.thnBuat << "\n\n";
        P = P->next;
    }
}
void insertLast(List &L, address P)
{
    if (!L.First)
        L.First = L.Last = P;
    else
    {
        L.Last->next = P;
        P->prev = L.Last;
        L.Last = P;
    }
}
address findElm(List L, string nopol)
{
    for (address P = L.First; P; P = P->next)
        if (P->info.nopol == nopol)
            return P;
    return NULL;
}
void deleteFirst(List &L, address &P)
{
    if (!L.First)
        return;
    P = L.First;
    if (L.First == L.Last)
        L.First = L.Last = NULL;
```

```
    else
    {
        L.First = L.First->next;
        L.First->prev = NULL;
    }
    P->next = NULL;
}
void deleteLast(List &L, address &P)
{
    if (!L.Last)
        return;
    P = L.Last;
    if (L.First == L.Last)
        L.First = L.Last = NULL;
    else
    {
        L.Last = L.Last->prev;
        L.Last->next = NULL;
    }
    P->prev = NULL;
}
void deleteAfter(address Prec, address &P)
{
    if (!Prec || !Prec->next)
        return;
    P = Prec->next;
    Prec->next = P->next;
    if (P->next)
        P->next->prev = Prec;
```

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

main2.cpp

```
#include "Doublylist.h"
#include "Doublylist.cpp"
int main()
{
    List L;
    CreateList(L);
    infotype x;
    string cari, hapus;
    address P, found;

    for (int i = 0; i < 3; i++)
    {
        cout << "Masukkan nomor polisi: ";
        cin >> x.nopol;
        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.thnBuat;

        if (findElm(L, x.nopol) != NULL)
        {
            cout << "Nomor polisi sudah terdaftar!\n";
        }
        else
        {
```

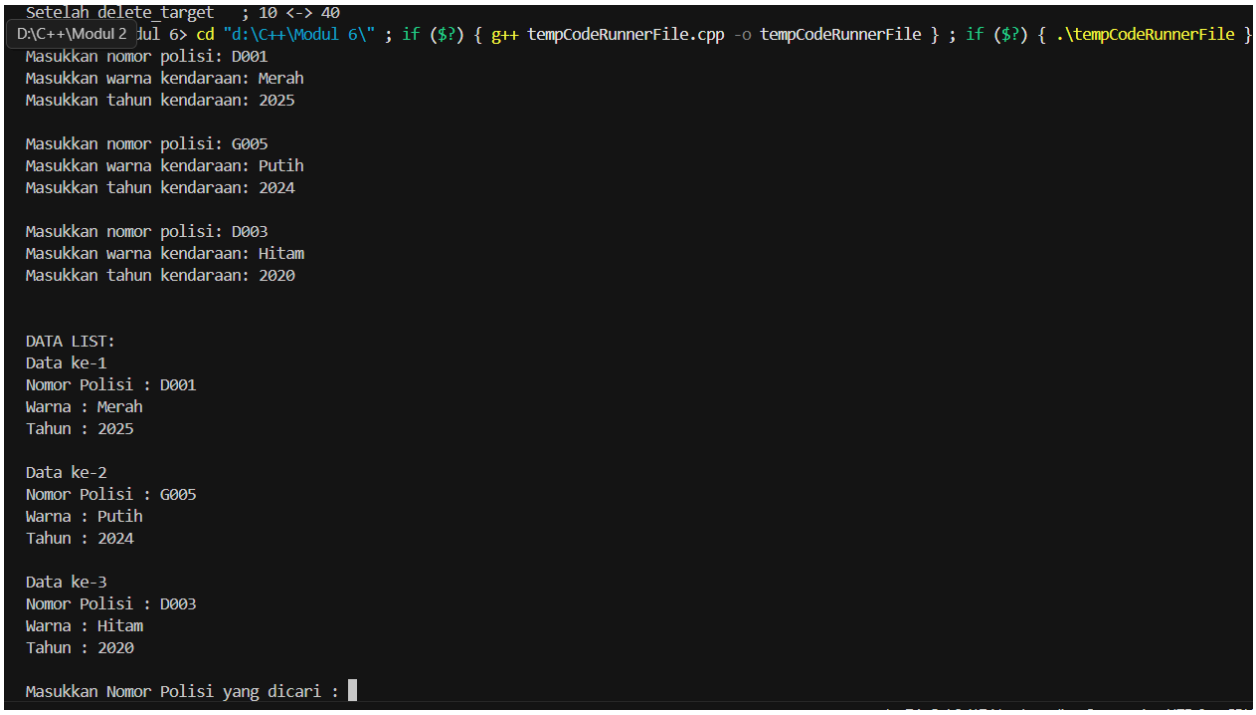
```
            P = alokasi(x);
            insertLast(L, P);
        }
        cout << endl;
    }
    printInfo(L);

    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> cari;
    found = findElm(L, cari);
    if (found != NULL)
    {
        cout << "\nData ditemukan:\n";
        cout << "Nomor Polisi : " << found->info.nopol << endl;
        cout << "Warna : " << found->info.warna << endl;
        cout << "Tahun : " << found->info.thnBuat << endl;
    }
    else
    {
        cout << "Data tidak ditemukan.\n";
    }

    cout << "\nMasukkan Nomor Polisi yang akan dihapus : ";
    cin >> hapus;
    found = findElm(L, hapus);
    if (found != NULL)
    {
        if (found == L.First)
        {
```

```
            deleteFirst(L, P);
        }
        else if (found == L.Last)
        {
            deleteLast(L, P);
        }
        else
        {
            deleteAfter(found->prev, P);
        }
        dealokasi(P);
            cout << "Data dengan nomor polisi " << hapus << "
berhasil dihapus.\n";
    }
    else
    {
        cout << "Data tidak ditemukan.\n";
    }
    printInfo(L);
    return 0;
}
```

Screenshoot Output



Deskripsi:

Program ini merupakan implementasi lengkap dari program manajemen data kendaraan menggunakan struktur data Doubly Linked List.

D. Kesimpulan

Praktikum ini berhasil mengimplementasikan struktur data Doubly Linked List (Daftar Berantai Ganda) menggunakan C++. Doubly Linked List dibedakan dari Singly Linked List karena setiap elemennya (node) memiliki dua penunjuk: next yang menunjuk ke elemen sesudah, dan prev yang menunjuk ke elemen sebelum. Implementasi ini mencakup komponen kunci seperti pointer first dan last untuk mengakses elemen di ujung list. Kehadiran penunjuk prev memungkinkan iterasi maju dan mundur pada list, secara signifikan mempermudah dan mengefisienkan operasi seperti Insert Before dan Delete Before. Secara keseluruhan, Doubly Linked List menawarkan fleksibilitas yang lebih besar dalam manipulasi elemen list dibandingkan Singly Linked List, terutama dalam operasi yang memerlukan akses ke node sebelumnya.

E. Referensi

https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170

https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd