LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 6
DOUBLY LINKED LIST

Disusun Oleh :
NAMA : M.IRFAN ADIB
NIM : 103112400257

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Double Linked List adalah salah satu struktur data dinamis yang terdiri dari rangkaian node, di mana setiap node memiliki tiga bagian utama, yaitu: data, pointer ke node sebelumnya (*prev*), dan pointer ke node berikutnya (*next*). Berbeda dengan Single Linked List yang hanya dapat ditelusuri satu arah, Double Linked List memungkinkan penelusuran data secara dua arah (maju dan mundur). Hal ini membuat proses seperti pencarian, penghapusan, atau penyisipan data menjadi lebih fleksibel karena kita bisa bergerak ke depan atau ke belakang sesuai kebutuhan. Double Linked List sangat berguna ketika diperlukan manipulasi data yang sering, terutama di tengah-tengah list, karena tidak membutuhkan pergeseran elemen seperti array. Namun, struktur ini juga memiliki kelemahan, yaitu penggunaan memori yang lebih besar karena setiap node menyimpan dua pointer. Meskipun demikian, kemampuannya yang lebih fleksibel dan efisien dalam menangani perubahan data menjadikan Double Linked List sebagai salah satu struktur data penting dalam pemrograman dan sistem komputer.

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_last(5);
    add_last(20);
    cout << "Awal\t\t\t\t\t : ";
    view();

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

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

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

    return 0;
}
```

Screenshots Output

```
Awal                    : 10 <-> 5 <-> 20
Setelah delete_first    : 5 <-> 20
Setelah delete_last     : 5
Setelah tambah          : 5 <-> 30 <-> 40
Setelah delete_target   : 5 <-> 40
```

Deskripsi:
Program di atas membuat sebuah Double Linked List, yaitu rangkaian data yang saling terhubung ke depan dan ke belakang, sehingga kita bisa bergerak maju atau mundur di dalam list. Program ini menyimpan dua penanda utama: bagian depan list dan bagian belakang list. Saat program berjalan, data pertama dimasukkan ke bagian depan, lalu data berikutnya ditaruh di belakang sehingga list mulai terbentuk seperti rantai. Setelah itu, program mencoba menghapus data paling depan, lalu menghapus data paling belakang, sehingga terlihat bagaimana bagian awal dan akhir list ikut berubah. Program juga menambah beberapa data baru lagi, menampilkannya, lalu menghapus data tertentu di tengah list. Melalui proses itu, kita bisa melihat bagaimana setiap data yang saling terhubung dua arah dapat ditambah, dihapus, dan ditata ulang hanya dengan mengatur hubungan antar node-nya tanpa harus memindahkan semua data.

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

Doublylist.h

```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;
```

```
typedef ElmList* address;

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

struct List {
    address First;
    address Last;
};

void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void insertLast(List &L, address P);
void printInfo(List L);
bool isExist(List L, string nopol);
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

```
#include "doublylist.h"

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

address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
```

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

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

bool isExist(List L, string nopol) {
    address P = L.First;
    while (P != NULL) {
        if (P->info.nopol == nopol) {
            return true;
        }
        P = P->next;
    }
    return false;
}

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;
}

void insertLast(List &L, address P) {
    if (L.First == NULL) {
        L.First = P;
        L.Last = P;
    } else {
        L.Last->next = P;
        P->prev = L.Last;
        L.Last = P;
    }
}
```

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

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

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;
}

void printInfo(List L) {
    address P = L.First;
    cout << "\nDATA LIST:\n";
    while (P != NULL) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna        : " << P->info.warna << endl;
        cout << "Tahun        : " << P->info.thnbuat << endl;
        cout << "-------------------------\n";
        P = P->next;
    }
}
```

main.cpp

```
#include "doublylist.h"
#include "doublylist.cpp"

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

    infotype x;
    char pilih = 'y';
    while (pilih == 'y' || pilih == 'Y') {
        cout << "Masukkan Nomor Polisi : ";
        cin >> x.nopol;
        cout << "Masukkan Warna        : ";
        cin >> x.warna;
        cout << "Masukkan Tahun        : ";
        cin >> x.thnbuat;

        if (isExist(L, x.nopol)) {
            cout << "Nomor polisi sudah terdaftar!\n";
        } else {
            address P = alokasi(x);
            insertLast(L, P);
        }

        cout << "Tambah data lagi? (y/n): ";
```

```
        cin >> pilih;
        cout << endl;
    }

    printInfo(L);

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

    string hapus;
    cout << "\nMasukkan Nomor Polisi yang akan dihapus : ";
    cin >> hapus;

    address target = findElm(L, hapus);
    if (target == NULL) {
        cout << "Data tidak ditemukan.\n";
    } else if (target == L.First) {
        address P;
        deleteFirst(L, P);
        dealokasi(P);
        cout << "Data pertama berhasil dihapus.\n";
    } else if (target == L.Last) {
        address P;
        deleteLast(L, P);
        dealokasi(P);
        cout << "Data terakhir berhasil dihapus.\n";
    } else {
        address Prec = target->prev;
        address P;
        deleteAfter(Prec, P);
        dealokasi(P);
        cout << "Data dengan nomor polisi " << hapus << " berhasil
dihapus.\n";
    }
```

```
    cout << "\nDATA SETELAH PENGHAPUSAN:\n";
    printInfo(L);

    return 0;
}
```

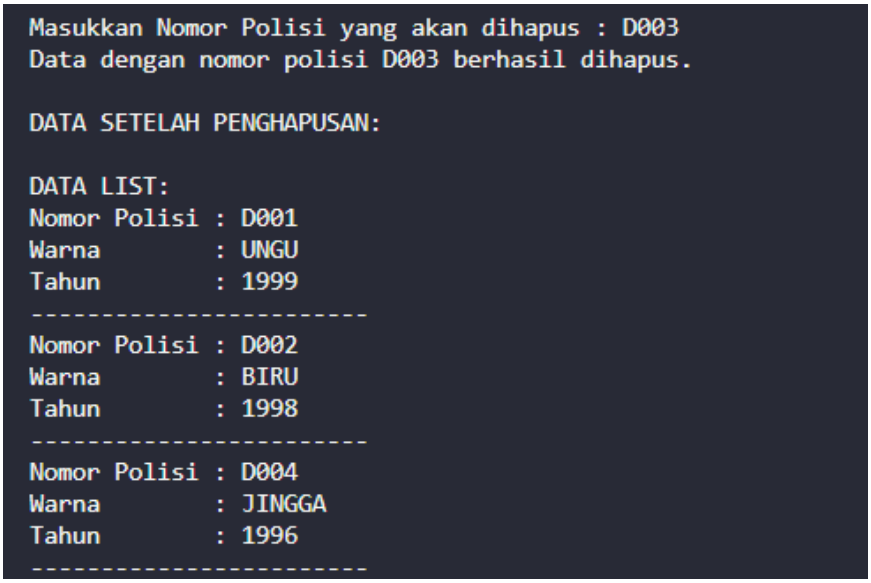
Screenshots Output 1

```
Masukkan Nomor Polisi : D001
Masukkan Warna        : UNGU
Masukkan Tahun        : 1999
Tambah data lagi? (y/n): Y

Masukkan Nomor Polisi : D002
Masukkan Warna        : BIRU
Masukkan Tahun        : 1998
Tambah data lagi? (y/n): Y

Masukkan Nomor Polisi : D003
Masukkan Warna        : KREM
Masukkan Tahun        : 1997
Tambah data lagi? (y/n): Y

Masukkan Nomor Polisi : D004
Masukkan Warna        : JINGGA
Masukkan Tahun        : 1996
Tambah data lagi? (y/n): N


DATA LIST:
Nomor Polisi : D001
Warna        : UNGU
Tahun        : 1999
-----------------------
Nomor Polisi : D002
Warna        : BIRU
Tahun        : 1998
-----------------------
Nomor Polisi : D003
Warna        : KREM
Tahun        : 1997
-----------------------
Nomor Polisi : D004
Warna        : JINGGA
Tahun        : 1996
-----------------------
```

Screenshots Output 2

```
Masukkan Nomor Polisi yang dicari : D001

Nomor Polisi : D001
Warna        : UNGU
Tahun        : 1999
```

Screenshots Output 3



Deskripsi:
Program ini bekerja untuk mengelola data kendaraan menggunakan doubly linked list, yaitu struktur data yang tiap elemennya saling terhubung ke depan dan ke belakang. Pertama, program menyiapkan list kosong, lalu menyediakan fungsi untuk membuat node baru berisi data kendaraan seperti nomor polisi, warna, dan tahun. Program juga bisa mengecek apakah nomor polisi tertentu sudah ada, mencari data kendaraan, menambahkan data ke bagian belakang list, serta menghapus data dari depan, belakang, atau setelah node tertentu. Setiap perubahan dilakukan dengan mengatur ulang sambungan antar node tanpa memindahkan seluruh data. Di akhir, program menyediakan fitur untuk menampilkan semua isi list dengan rapi. keseluruhan kode ini bertujuan agar data bisa ditambah, dicari, ditampilkan, dan dihapus dengan lebih mudah dan teratur.

D. Kesimpulan

Kesimpulannya, program ini membantu kita mengelola data kendaraan dengan cara yang rapi dan fleksibel menggunakan doubly linked list. Dengan struktur ini, kita bisa menambah, mencari, menampilkan, dan menghapus data dari berbagai posisi tanpa harus memindahkan seluruh elemen. Setiap node saling terhubung dua arah, sehingga proses navigasi dan pengeditan data jadi lebih mudah. Secara keseluruhan, program ini menunjukkan cara kerja struktur data yang efisien untuk menangani data yang berubah-ubah dan butuh pengelolaan dinamis

E. Referensi

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). *Introduction to Algorithms* (4th ed.). The MIT Press.

Stroustrup, B. (2014). *Programming: Principles and Practice Using C++* (2nd ed.). Addison-Wesley Professional.