LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 6
DOUBLY LINKED LIST

Disusun Oleh :
NAMA : Muhammad Omar Nadiv
NIM : 103112430063

Dosen
Fahrudin Mukti Wibowo, S.Kom., M.Eng.

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Doubly linked list (DLL) adalah struktur data linier di mana setiap node memiliki dua penunjuk (pointer), yaitu satu menunjuk ke node sebelumnya (prev) dan satu lagi menunjuk ke node berikutnya (next). Struktur ini memungkinkan proses traversi data dilakukan dua arah, baik dari awal ke akhir maupun dari akhir ke awal.

Keunggulan Doubly Linked List dibandingkan dengan Singly Linked List adalah kemampuannya untuk melakukan operasi penyisipan dan penghapusan dengan lebih efisien, karena tidak perlu melakukan traversal satu arah untuk menemukan node sebelumnya. Setiap node saling terhubung secara dua arah, sehingga pengelolaan data menjadi lebih fleksibel.

Operasi yang umum dilakukan pada Doubly Linked List meliputi penambahan data (insert) di awal, akhir, atau setelah node tertentu; penghapusan data (delete) di berbagai posisi; serta pencarian (search) dan penampilan data (view). Dengan menggunakan pointer ganda (prev dan next), struktur ini juga lebih mudah diatur untuk aplikasi yang memerlukan navigasi bolak-balik antar data.

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);
```

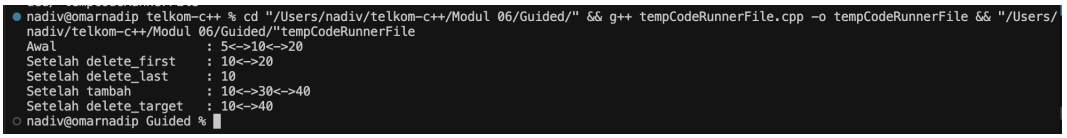
```
        }
        else
        {
            Node *newNode = new Node{newValue, current, current->next};
            current->next->prev = newNode;
            current->next = newNode;
        }
    }
}

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;
        }
        else if (current == ptr_last)
```

```
        {
            delete_last();
            return;
        }
        else
        {
            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 digunakan untuk mengelola data dalam struktur data Doubly Linked List menggunakan bahasa C++. Struct Node menyimpan data berupa nilai integer serta dua pointer, yaitu prev untuk menunjuk node sebelumnya dan next untuk menunjuk node berikutnya. Program ini mendukung berbagai operasi dasar seperti menambah data di awal (add_first), di akhir (add_last), dan setelah nilai tertentu (add_target).

Selain itu, program juga menyediakan fungsi untuk menghapus node di awal (delete_first), di akhir (delete_last), maupun berdasarkan nilai tertentu (delete_target), serta mengedit data node melalui fungsi edit_node. Fungsi view digunakan untuk menampilkan isi list secara berurutan dari depan ke belakang dengan simbol penghubung <->.

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

Unguided 1 (file 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;
typedef struct 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 printInfo(List L);
void insertLast(List &L, address P);
address findElm(List L, string nopol);
void deleteFirst(List &L);
void deleteLast(List &L);
void deleteAfter(List &L, address prec);
```

```
#endif
```

Unguided 1 (file 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;
}

void printInfo(List L)
{
   if (L.first == NULL)
   {
       cout << "List kosong." << endl;
       return;
   }
   address P = L.first;
   while (P != NULL)
   {
       cout << P->info.nopol << " | " << P->info.warna
            << " | " << P->info.thnBuat << endl;
       P = P->next;
```

```
    }
}

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;
    }
}

address findElm(List L, string nopol)
{
    address P = L.first;
    while (P != NULL && P->info.nopol != nopol)
    {
        P = P->next;
    }
    return P;
}

void deleteFirst(List &L)
{
    if (L.first == NULL)
        return;
    address P = L.first;
    if (L.first == L.last)
    {
        L.first = NULL;
        L.last = NULL;
    }
    else
    {
        L.first = P->next;
```

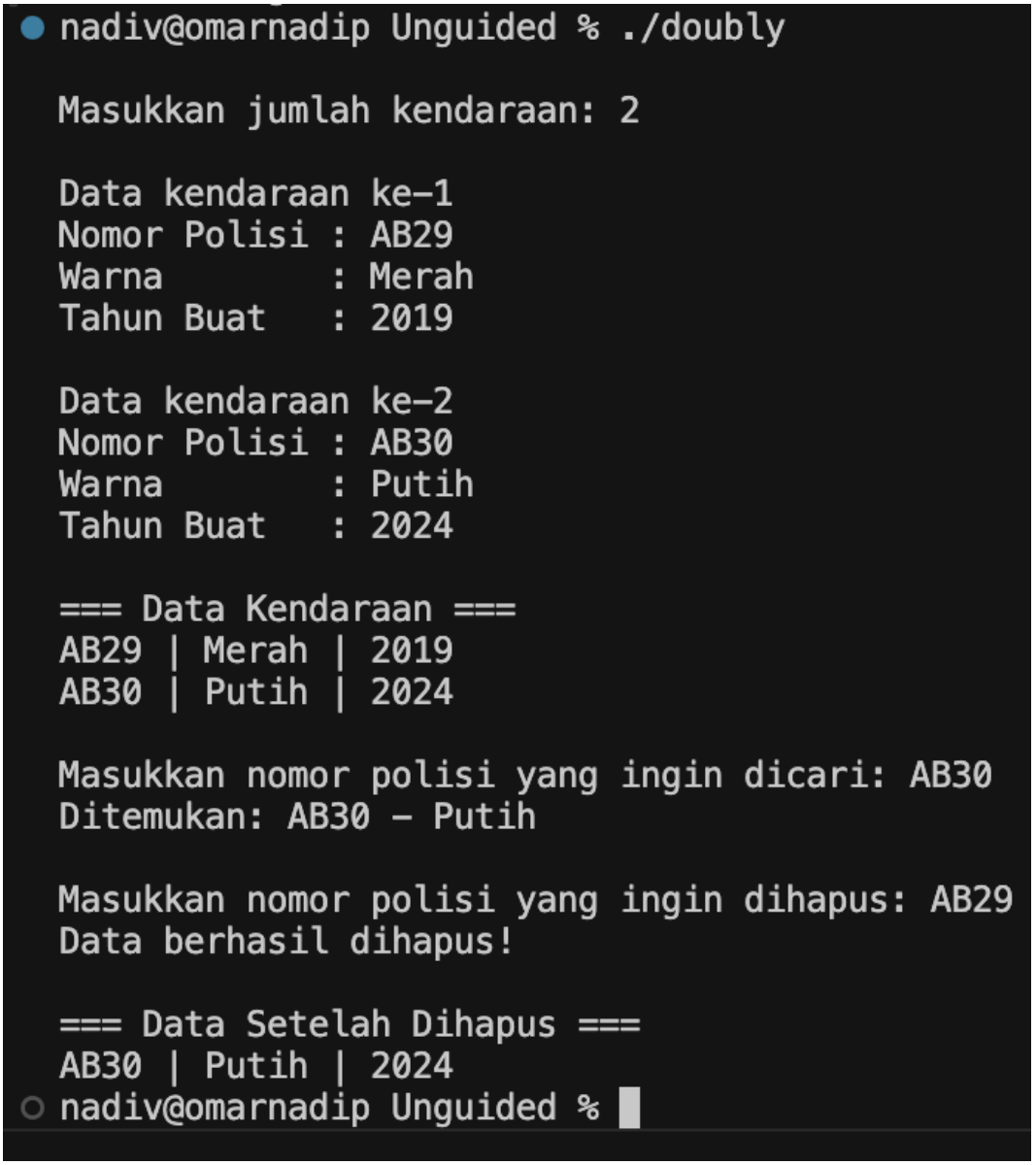
```
        L.first->prev = NULL;
    }
    dealokasi(P);
}

void deleteLast(List &L)
{
    if (L.last == NULL)
        return;
    address P = L.last;
    if (L.first == L.last)
    {
        L.first = NULL;
        L.last = NULL;
    }
    else
    {
        L.last = P->prev;
        L.last->next = NULL;
    }
    dealokasi(P);
}

void deleteAfter(List &L, address prec)
{
    if (prec == NULL || prec->next == NULL)
        return;
    address P = prec->next;
    prec->next = P->next;
    if (P->next != NULL)
    {
        P->next->prev = prec;
    }
    else
    {
        L.last = prec;
    }
    dealokasi(P);
}
```

Unguided 1 (file main.cpp)

```
#include "Doublylist.h"

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

   int n;
   cout << "Masukkan jumlah kendaraan: ";
   cin >> n;
   cin.ignore();

   for (int i = 0; i < n; i++)
   {
       Kendaraan k;
       cout << "\nData kendaraan ke-" << i + 1 << endl;
       cout << "Nomor Polisi : ";
       getline(cin, k.nopol);
       cout << "Warna        : ";
       getline(cin, k.warna);
       cout << "Tahun Buat   : ";
       cin >> k.thnBuat;
       cin.ignore();

       insertLast(L, alokasi(k));
   }

   cout << "\n=== Data Kendaraan ===" << endl;
   printInfo(L);

   string cari;
   cout << "\nMasukkan nomor polisi yang ingin dicari: ";
   getline(cin, cari);
   address found = findElm(L, cari);
   if (found != NULL)
       cout << "Ditemukan: " << found->info.nopol << " - " <<
found->info.warna << endl;
   else
       cout << "Data tidak ditemukan." << endl;
```

```
    string hapus;
    cout << "\nMasukkan nomor polisi yang ingin dihapus: ";
    getline(cin, hapus);
    address target = findElm(L, hapus);
    if (target != NULL)
    {
        if (target == L.first)
            deleteFirst(L);
        else if (target == L.last)
            deleteLast(L);
        else
            deleteAfter(L, target->prev);
        cout << "Data berhasil dihapus!" << endl;
    }
    else
    {
        cout << "Data tidak ditemukan!" << endl;
    }

    cout << "\n=== Data Setelah Dihapus ===" << endl;
    printInfo(L);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini digunakan untuk mengelola data kendaraan menggunakan struktur data Doubly Linked List dalam bahasa C++. Struct Kendaraan menyimpan informasi berupa nomor polisi, warna, dan tahun pembuatan. Setiap data kendaraan disimpan dalam node yang memiliki dua pointer, yaitu prev untuk menunjuk elemen sebelumnya dan next untuk menunjuk elemen berikutnya.

Program meminta pengguna untuk memasukkan jumlah kendaraan, kemudian menginput data setiap kendaraan satu per satu. Data yang

dimasukkan akan ditambahkan ke dalam list menggunakan fungsi insertLast. Selain itu, program juga menyediakan fitur untuk mencari kendaraan berdasarkan nomor polisi, menghapus data kendaraan tertentu, dan menampilkan seluruh data kendaraan yang tersimpan dalam list.

D. Kesimpulan

Pada modul 6 ini, saya belajar bagaimana menggunakan struktur data Doubly Linked List dalam pemrograman C++. Saya memahami cara membuat struct dengan dua pointer (prev dan next) untuk menghubungkan node secara dua arah, serta cara menambah, menghapus, mencari, dan menampilkan data dalam list. Selain itu, saya juga belajar bagaimana mengelola memori secara dinamis dan memahami perbedaan antara singly dan doubly linked list. Pembelajaran ini membantu saya memahami konsep struktur data yang lebih kompleks, efisien, dan fleksibel dalam pengolahan data secara dua arah.

E. Referensi

https://www.geeksforgeeks.org/dsa/doubly-linked-list/

https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html

https://dev.to/emmanuelayinde/master-how-doubly-linked-list-is-implemented-in-javascript-ogi