

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 6
DOUBLY LINKED LIST
(BAGIAN PERTAMA)**



Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO**

2025

A. Dasar Teori

Doubly Linked List adalah struktur data linear di mana setiap elemen (node) memiliki dua penunjuk, yaitu prev yang menunjuk ke elemen sebelumnya dan next yang menunjuk ke elemen sesudahnya. Struktur ini memungkinkan navigasi dua arah, maju dan mundur, melalui daftar elemen. Setiap Doubly Linked List memiliki dua penunjuk utama: first yang menunjuk ke elemen pertama dan last yang menunjuk ke elemen terakhir dalam daftar. Komponen penting lainnya termasuk info (data yang disimpan dalam elemen), next (penunjuk ke elemen berikutnya), dan prev (penunjuk ke elemen sebelumnya).

Operasi dasar pada Doubly Linked List meliputi insert (menambah elemen di awal, akhir, atau di antara elemen lain) dan delete (menghapus elemen di awal, akhir, atau di antara elemen lain) yang dapat dilakukan dengan efisien. Selain itu, proses pencarian, pembaruan, dan tampilan data pada Doubly Linked List lebih mudah dibandingkan Singly Linked List karena kemampuan iterasi maju dan mundur.

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

“main.cpp”

```
#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;
}

```

Screenshots Output

```

PS D:\103112430054_Lahra Budi Saputra_IF 12-06> cd "d:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Guided)\"; 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:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Guided)>

```

Deskripsi:

Program di atas merupakan implementasi praktis dari struktur data ***Doubly Linked List (DLL)***, yang konsepnya sejalan dengan materi pada modul. Program ini mendefinisikan sebuah *struct Node* yang berfungsi sebagai elemen list, berisi int data (sebagai infotype), serta pointer Node prev dan Node next untuk navigasi dua arah. Berbeda dengan modul yang membungkus pointer list dalam struct list, implementasi ini menggunakan dua pointer global, ptr_first dan ptr_last, untuk melacak elemen pertama dan terakhir.

Fungsionalitas inti DLL diimplementasikan melalui beberapa fungsi: add_first dan add_last menangani penyisipan di awal dan akhir list (Insert First & Last), sementara add_target berfungsi sebagai Insert After untuk menyisipkan node baru setelah nilai target ditemukan.

Demikian pula, operasi penghapusan ditangani oleh delete_first dan delete_last (Delete First & Last), serta delete_target yang mencari dan menghapus node dengan nilai tertentu. Semua fungsi ini mengelola pointer prev dan next dengan cermat, termasuk menangani kasus-kasus khusus seperti list kosong atau list yang hanya memiliki satu elemen. Kode ini juga menyertakan fungsi utilitas view untuk mencetak

seluruh isi list (setara printInfo) dan edit_node untuk memperbarui nilai node (Update). Fungsi main bertindak sebagai driver yang menguji berbagai operasi ini secara berurutan, seperti menambah, menghapus, dan kemudian menampilkan hasil list untuk memverifikasi bahwa semua operasi berjalan sesuai harapan.

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

Unguided Soal 1

(“Doublylist.h”:)

```
#ifndef DOUBLYLIST_H

#define DOUBLYLIST_H

#include <iostream>

#include <string>

#include <algorithm>

struct kendaraan {

    std::string nopol; // Nomor Polisi

    std::string warna;

    int thnBuat; // Tahun Pembuatan

};

typedef struct ElmList* address;

struct ElmList {

    kendaraan info;
```

```
address next;

address prev;

};

struct List {

address First;

address Last;

};

// procedure CreateList

void CreateList(List &L);

// function alokasi

address alokasi(kendaraan x);

// procedure dealokasi

void dealokasi(address &P);

// procedure printInfo

void printInfo(List L);

// procedure insertLast

void insertLast(List &L, address P);

// function search

address search(List L, std::string nopol_target);

#endif
```

(“Doublylist.cpp”)

```
#include "Doublylist.h"

void CreateList(List &L) {

    L.First = NULL;

    L.Last = NULL;

}

address alokasi(kendaraan x) {

    address P = new ElmList;

    if (P != NULL) {

        P->info = x;

        P->next = NULL;

        P->prev = NULL;

    }

    return P;

}

void dealokasi(address &P) {

    delete P;

    P = NULL;

}

void insertLast(List &L, address P) {

    if (P == NULL) {
```

```
return;

}

if (L.First == NULL) {

    L.First = P;

    L.Last = P;

} else {

    L.Last->next = P;

    P->prev = L.Last;

    L.Last = P;

}

P->next = NULL;

}

address search(List L, std::string nopol_target) {

    address P = L.First;

    while (P != NULL) {

        if (P->info.nopol == nopol_target) {

            return P;

        }

        P = P->next;

    }

    return NULL;

}
```

```

}

// Buat ngecetak list secara mundur

void printInfo(List L) {

if (L.First == NULL) {

std::cout << "List kosong." << std::endl;

return;

}

address P = L.Last; // Mulai dari Last

std::cout << "\nDATA LIST 1" << std::endl;

while (P != NULL) {

std::cout << "no polisi : " << P->info.nopol << std::endl;

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

std::cout << "tahun : " << P->info.thnBuat << std::endl;

P = P->prev;

}

}

```

(main.cpp)

```

#include "Doublylist.h"

void processInput(List &L, std::string nopol, std::string warna, int thnBuat) {

kendaraan k;

```

```

k.nopol = nopol;

k.warna = warna;

k.thnBuat = thnBuat;

// Buat Ngecetak semua baris input

std::cout << "masukkan nomor polisi: " << nopol << std::endl;

std::cout << "masukkan warna kendaraan: " << warna << std::endl;

std::cout << "masukkan tahun kendaraan: " << thnBuat << std::endl;

// Buat ngecek duplikasi dan insert

if (search(L, k.nopol) == NULL) {

address P = alokasi(k);

if (P) {

insertLast(L, P);

}

} else {

std::cout << "nomor polisi sudah terdaftar" << std::endl << std::flush;

}

std::cout << std::endl; // Pemisah antar input

}

int main() {

List L;

CreateList(L);

```

```

// Input D001 (Berhasil)

processInput(L, "D001", "hitam", 90);

// Input D003 (Berhasil)

processInput(L, "D003", "putih", 70);

// Input D001 (Gagal, Duplikat)

processInput(L, "D001", "merah", 80);

// Input D004 (Berhasil)

processInput(L, "D004", "kuning", 90);

printInfo(L);

return 0;

}

```

Screenshots Output:

```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)> g++ Doublylist.cpp main.cpp -o DoublyListTest
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)> ./DoublyListTest
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
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)>

```

Deskripsi:

Ketiga file tersebut bekerja bersama untuk mengimplementasikan Abstract Data Type (ADT) Doubly Linked List (DLL) yang dirinci dalam Modul 6, khusus untuk mengelola data kendaraan. **File Doublylist.h** bertindak sebagai "cetak biru" (header), yang mendefinisikan struktur data inti: struct kendaraan sebagai infotype, struct ElmList sebagai node yang berisi pointer next dan prev, serta struct List sebagai manajer list yang memiliki pointer First dan Last. File ini juga mendeklarasikan semua prototipe fungsi yang diperlukan, seperti CreateList, alokasi, insertLast, search, dan printInfo.

Kemudian, **file Doublylist.cpp** berfungsi sebagai "mesin" atau file implementasi, yang menyediakan kode C++ dan logika untuk setiap fungsi yang dideklarasikan di header. Implementasi kuncinya mencakup insertLast yang menangani penambahan di akhir list dan search yang mencari nopol dari awal. Fungsi printInfo secara khusus mengimplementasikan penelusuran mundur (dari L.Last ke L.First menggunakan prev), sebuah keunggulan DLL, untuk mencocokkan format output pada contoh modul. **Terakhir, main.cpp** adalah file "driver" yang menjalankan program; file ini menggunakan fungsi pembantu processInput untuk mensimulasikan empat skenario input dari contoh, lengkap dengan pencetakan prompt, pengecekan duplikat nopol menggunakan search(), dan akhirnya memanggil printInfo untuk menampilkan hasil akhir list.

Unguided Soal 2

Hanya Mengganti / Menambahkan Fungsi (baru) main.cpp

```
#include "Doublylist.h"

// Fungsi processInput tetap sama

void processInput(List &L, std::string nopol, std::string warna, int thnBuat) {

    kendaraan k;

    k.nopol = nopol;
```



```

k.warna = warna;

k.thnBuat = thnBuat;

// Buat Ngecetak semua baris input

std::cout << "masukkan nomor polisi: " << nopol << std::endl;

std::cout << "masukkan warna kendaraan: " << warna << std::endl;

std::cout << "masukkan tahun kendaraan: " << thnBuat << std::endl;

// Buat ngecek duplikasi dan insert

if (search(L, k.nopol) == NULL) {

address P = alokasi(k);

if (P) {

insertLast(L, P);

}

} else {

std::cout << "nomor polisi sudah terdaftar" << std::endl << std::flush;

}

std::cout << std::endl; // Pemisah antar input

}

int main() {

List L;

CreateList(L);

// === Latihan 1: Memasukkan Data ===

```

```

std::cout << "===> Latihan 1: Input Data <===" << std::endl;

processInput(L, "D001", "hitam", 90);

processInput(L, "D003", "putih", 70);

processInput(L, "D001", "merah", 80);

processInput(L, "D004", "kuning", 90);

// Cetak hasil Latihan 1

printInfo(L);

// === Latihan 2: Mencari Data ===

std::cout << "\n===> Latihan 2: Mencari Data <===" << std::endl;

std::string nopol_dicari;

// Prompt sesuai gambar [cite: 415]

std::cout << "Masukkan Nomor Polisi yang dicari : ";

std::cin >> nopol_dicari;

// Panggil fungsi search (findElm)

address found = search(L, nopol_dicari);

if (found != NULL) {

// Tampilkan hasil sesuai gambar [cite: 416-418]

std::cout << "Nomor Polisi \t: " << found->info.nopol << std::endl;

std::cout << "Warna \t\t: " << found->info.warna << std::endl;

std::cout << "Tahun \t\t: " << found->info.thnBuat << std::endl;

} else {

```

```

std::cout << "Data dengan nomor polisi " << nopol_dicari << " tidak ditemukan."
<< std::endl;

}

return 0;

}

```

Screenshots Output:

```

PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)> g++ Doublylist.cpp main.cpp -o DoublyListTest
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)> ./DoublyListTest
===> Latihan 1: Input Data <===
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

===> Latihan 2: Mencari Data <===
Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi      : D001
Warna              : hitam
Tahun              : 90
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 (Unguided)>

```

Deskripsi:

Kode ini adalah implementasi Doubly Linked List (DLL) yang dipisah menjadi tiga file (.h, .cpp, main.cpp) untuk mengelola data kendaraan. File main.cpp menjalankan dua tugas: pertama, ia mensimulasikan **Latihan 1** dengan menambahkan data kendaraan ("D001", "D003", "D004"), sambil menggunakan fungsi search untuk menolak input duplikat ("D001" kedua). Setelah itu, ia menjalankan **Latihan 2** dengan meminta pengguna memasukkan nomor polisi yang ingin dicari. Program kemudian memanggil fungsi search untuk menemukan data tersebut ("D001") dan mencetak detail lengkapnya (warna dan tahun) ke layar, sesuai dengan output yang diminta.

Unguided Soal 3

(File Header (Doublylist.h)

```
#ifndef DOUBLYLIST_H

#define DOUBLYLIST_H

#include <iostream>

#include <string>

#include <algorithm>

struct kendaraan {

    std::string nopol; // Nomor Polisi

    std::string warna;

    int thnBuat; // Tahun Pembuatan

};

typedef struct ElmList* address;

struct ElmList {

    kendaraan info;

    address next;

    address prev;

};

struct List {

    address First;

    address Last;
```

```

};

void CreateList(List &L);

address alokasi(kendaraan x);

void dealokasi(address &P);

void printInfo(List L);

void insertLast(List &L, address P);

address search(List L, std::string nopol_target);

void deleteFirst(List &L, address &P);

void deleteLast(List &L, address &P);

void deleteAfter(List &L, address Prec, address &P);

#endif

```

(File Implementasi (Doublylist.cpp))

```

#include "Doublylist.h"

void CreateList(List &L) {

    L.First = NULL;

    L.Last = NULL;

}

address alokasi(kendaraan x) {

    address P = new ElmList;

    if (P != NULL) {

```

```

P->info = x;

P->next = NULL;

P->prev = NULL;

}

return P;

}

void dealokasi(address &P) {

delete P;

P = NULL;

}

void insertLast(List &L, address P) {

if (P == NULL) {

return;

}

if (L.First == NULL) {

L.First = P;

L.Last = P;

} else {

L.Last->next = P;

P->prev = L.Last;

L.Last = P;

```

```

}

P->next = NULL;

}

address search(List L, std::string nopol_target) {

address P = L.First;

while (P != NULL) {

if (P->info.nopol == nopol_target) {

return P;

}

P = P->next;

}

return NULL;

}

void printInfo(List L) {

if (L.First == NULL) {

std::cout << "List kosong." << std::endl;

return;

}

address P = L.Last; // Mulai dari Last

std::cout << "\nDATA LIST I" << std::endl;

while (P != NULL) {

```

```
std::cout << "no polisi : " << P->info.nopol << std::endl;
```

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

```
std::cout << "tahun : " << P->info.thnBuat << std::endl;
```

```
P = P->prev;
```

```
}
```

```
}
```

```
// IMPLEMENTASI BARU UNTUK LATIHAN 3
```

```
void deleteFirst(List &L, address &P) {
```

```
if (L.First == NULL) {
```

```
P = NULL; // List kosong
```

```
return;
```

```
}
```

```
P = L.First; // Simpan elemen pertama
```

```
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(List &L, address Prec, address &P) {

if (Prec == NULL || Prec->next == NULL) {

P = NULL;

return;

}

```

```

P = Prec->next; // Simpan elemen yang akan dihapus

if (P == L.Last) {

deleteLast(L, P);

} else {

Prec->next = P->next;

P->next->prev = Prec;

P->next = NULL;

P->prev = NULL;

}

}

```

(File Utama (main.cpp)

```

#include "Doublylist.h"

void processInput(List &L, std::string nopol, std::string warna, int thnBuat) {

kendaraan k;

k.nopol = nopol;

k.warna = warna;

k.thnBuat = thnBuat;

std::cout << "masukkan nomor polisi: " << nopol << std::endl;

std::cout << "masukkan warna kendaraan: " << warna << std::endl;

std::cout << "masukkan tahun kendaraan: " << thnBuat << std::endl;

```

```

if (search(L, k.nopol) == NULL) {

    address P = alokasi(k);

    if (P) {

        insertLast(L, P);

    }

} else {

    std::cout << "nomor polisi sudah terdaftar" << std::endl << std::flush;

}

std::cout << std::endl;

}

int main() {

    List L;

    CreateList(L);

    // ==> Latihan 1: Memasukkan Data <==

    std::cout << "==" << "Latihan 1: Input Data <==" << std::endl;

    processInput(L, "D001", "hitam", 90);

    processInput(L, "D003", "putih", 70);

    processInput(L, "D001", "merah", 80);

    processInput(L, "D004", "kuning", 90);

    // ==> Latihan 2: Mencari Data <==

    std::cout << "\n==" << "Latihan 2: Mencari Data <==" << std::endl;

```

```

std::string nopol_dicari;

std::cout << "Masukkan Nomor Polisi yang dicari : D001" << std::endl;

nopol_dicari = "D001";

address found = search(L, nopol_dicari);

if (found != NULL) {

std::cout << "Nomor Polisi \t: " << found->info.nopol << std::endl;

std::cout << "Warna \t\t: " << found->info.warna << std::endl;

std::cout << "Tahun \t\t: " << found->info.thnBuat << std::endl;

} else {

std::cout << "Data dengan nomor polisi " << nopol_dicari << " tidak ditemukan."
<< std::endl;

}

// ==> LATIHAN 3: MENGHAPUS DATA <==

std::cout << "\n==> Latihan 3: Menghapus Data <==" << std::endl;

std::string nopol_hapus;

std::cout << "Masukkan Nomor Polisi yang akan dihapus : D003" << std::endl;

nopol_hapus = "D003";

address P_hapus = NULL;

address P_target = search(L, nopol_hapus);

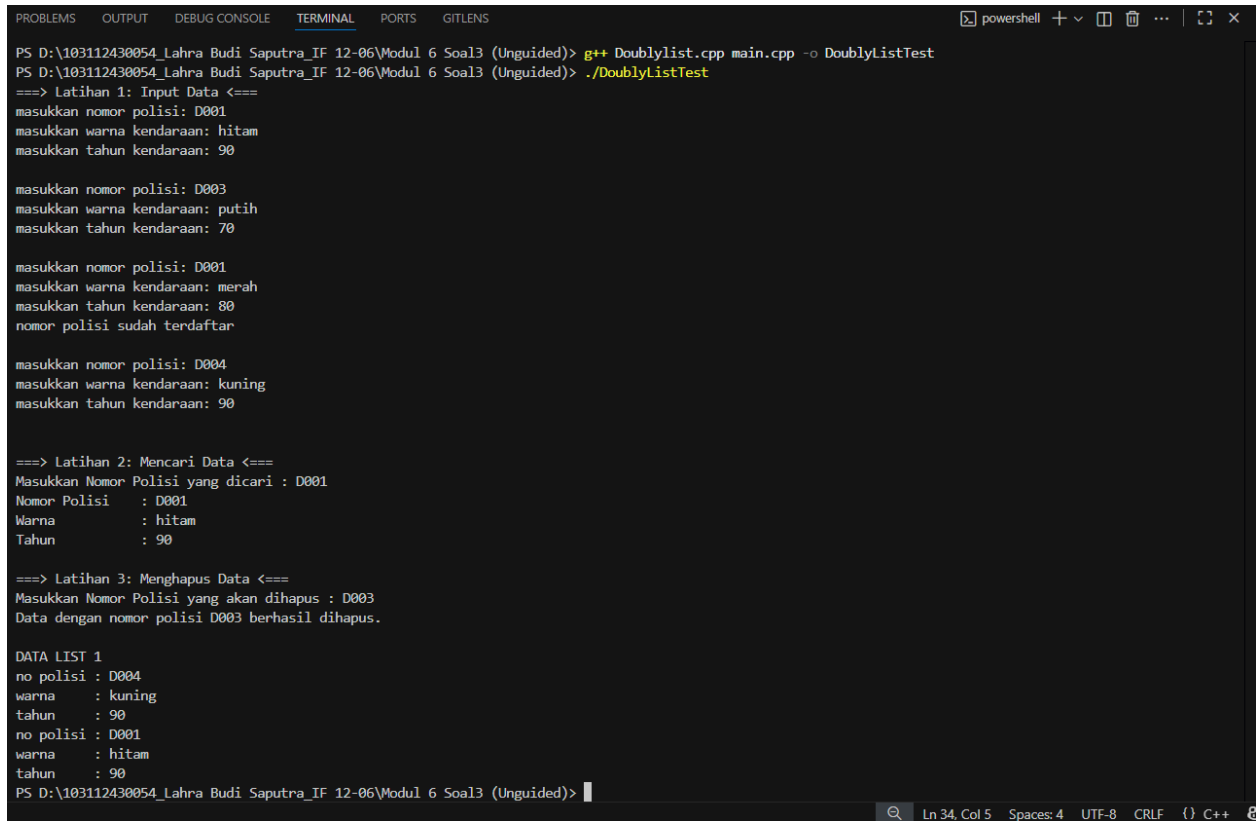
if (P_target == NULL) {

std::cout << "Data dengan nomor polisi " << nopol_hapus << " tidak ditemukan."
<< std::endl;

```

```
} else {  
  
    // Tentukan prosedur delete yang tepat  
  
    if (P_target == L.First) {  
  
        deleteFirst(L, P_hapus);  
  
    } else if (P_target == L.Last) {  
  
        deleteLast(L, P_hapus);  
  
    } else {  
  
        deleteAfter(L, P_target->prev, P_hapus);  
  
    }  
  
    // Dealokasi node yang dikembalikan  
  
    dealokasi(P_hapus);  
  
    // Cetak pesan sukses  
  
    std::cout << "Data dengan nomor polisi " << nopol_hapus << " berhasil dihapus."  
    << std::endl;  
  
}  
  
    // Tampilkan list terakhir  
  
    printInfo(L);  
  
    return 0;  
  
}
```

Screenshots Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 Soal3 (Unguided)> g++ Doublylist.cpp main.cpp -o DoublyListTest
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 Soal3 (Unguided)> ./DoublyListTest

===> Latihan 1: Input Data <===
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

===> Latihan 2: Mencari Data <===
Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi : D001
Warna : hitam
Tahun : 90

===> Latihan 3: Menghapus Data <===
Masukkan Nomor Polisi yang akan dihapus : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1
no polisi : D004
warna : kuning
tahun : 90
no polisi : D001
warna : hitam
tahun : 90
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 6 Soal3 (Unguided)>
```

Deskripsi:

Kode ini adalah implementasi **Doubly Linked List (DLL)** yang mencakup Latihan 1, 2, dan 3 dari modul, yang dipisahkan menjadi tiga file ADT (.h, .cpp, main.cpp). **File .h dan .cpp** mendefinisikan semua struktur data (kendaraan, List) dan fungsi yang diperlukan, termasuk *insertLast*, *search*, *deleteFirst*, *deleteLast*, dan *deleteAfter*. **File main.cpp** bertindak sebagai driver yang mensimulasikan semua skenario: ia memasukkan data (menangani duplikat), mencari data "D001", dan menghapus data "D003". Akhirnya, ia mencetak list final yang tersisa menggunakan *printInfo*.

D. Kesimpulan

Praktikum Modul ini mendemonstrasikan konsep dan implementasi struktur data Doubly Linked List (DLL). Pembelajaran utama dari modul ini adalah pemahaman bahwa setiap elemen (node) pada DLL memiliki dua pointer next yang menunjuk ke elemen sesudahnya dan prev yang menunjuk ke elemen sebelumnya. Fitur ini memberikan keunggulan fundamental dibandingkan Singly Linked List, yaitu kemampuan untuk melakukan navigasi dua arah (maju dan mundur), yang membuat proses seperti pencarian

dan pencetakan data menjadi lebih fleksibel.

Pada bagian Guided, konsep-konsep dasar DLL diimplementasikan menggunakan pointer global (`ptr_first` dan `ptr_last`). Latihan ini mencakup implementasi fungsionalitas inti seperti penyisipan (`add_first`, `add_last`, `add_target`) dan penghapusan (`delete_first`, `delete_last`, `delete_target`), serta penanganan kasus-kasus khusus seperti list kosong atau list dengan satu elemen.

Pada bagian Unguided, konsep tersebut dikembangkan menjadi Abstract Data Type (ADT) yang lebih rapi dengan memisahkan deklarasi (.h) dan implementasi (.cpp). Dalam studi kasus data kendaraan, fungsi `search` berhasil diterapkan untuk memvalidasi input dan mencegah duplikasi data (nomor polisi). Keunggulan pointer `prev` dibuktikan secara praktis melalui fungsi `printInfo` yang mencetak seluruh isi list dengan cara menelusuri secara mundur, yaitu dari `L.Last` menuju `L.First`, sehingga menghasilkan output yang sesuai dengan permintaan modul.

E. Referensi

- [1] Computerphile. 2018. *Arrays vs Linked Lists - Computerphile*.
- [2] Dat Hoang. 2018. *Performance of Array vs. Linked-List on Modern Computers*.
<https://dzone.com/articles/performance-of-array-vs-linked-list-on-modern-comp>
- [3] Agung Wijoyo, Lalu Akbar Prayudi, Muhamad Fiqih, Rendi Dwi Santoso, Ricky Tri Setiawan Putra, Teguh Arifin, & Ahmad Farhan. (2024). Penggunaan Algoritma Doubly Linked List Untuk Insertion Dan Deletion. *JRIIN :Jurnal Riset Informatika Dan Inovasi*, 1(12), 1329–1331. Retrieved from <https://jurnalmahasiswa.com/index.php/jriin/article/view/1282>