LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLY LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :
Muhammad Fathammubina
NIM : 10 3112430188

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

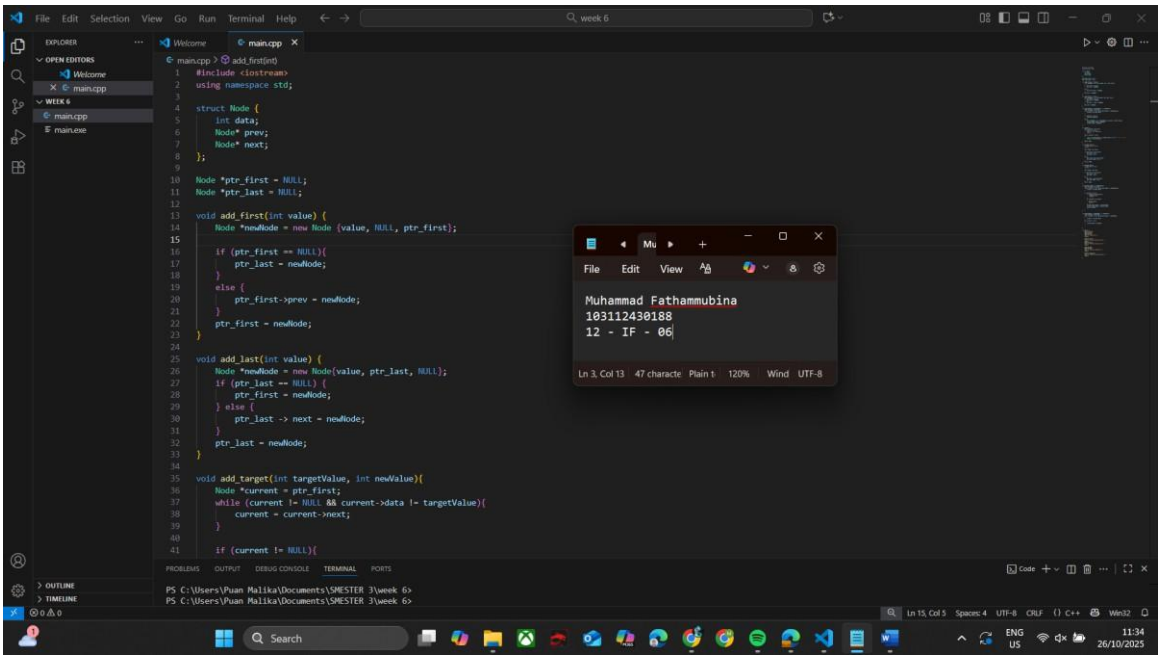
A. Dasar Teori

Doubly Linked List merupakan salah satu struktur data dinamis yang memungkinkan setiap elemen (node) memiliki dua pointer: satu menunjuk ke elemen sebelumnya (prev) dan satu menunjuk ke elemen sesudahnya (next). Berbeda dengan singly linked list yang hanya bisa diakses satu arah, doubly linked list memungkinkan traversal dua arah (maju dan mundur), sehingga lebih fleksibel dalam operasi seperti penyisipan (insert), penghapusan (delete), dan pencarian (search). Komponen utama dalam struktur ini meliputi pointer First yang menunjuk ke elemen pertama, Last yang menunjuk ke elemen terakhir, serta Prev dan Next yang menghubungkan antar elemen. Dengan konsep ADT (Abstract Data Type), setiap operasi seperti insertFirst, insertLast, deleteFirst, deleteLast, dan deleteAfter dapat diimplementasikan secara modular menggunakan bahasa C++. Prinsip kerja doubly linked list memungkinkan efisiensi dalam manipulasi data karena tidak memerlukan pergeseran elemen seperti pada array.

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

Guided 1

Screenshoot Program




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

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

Node *ptr_first = NULL;
```

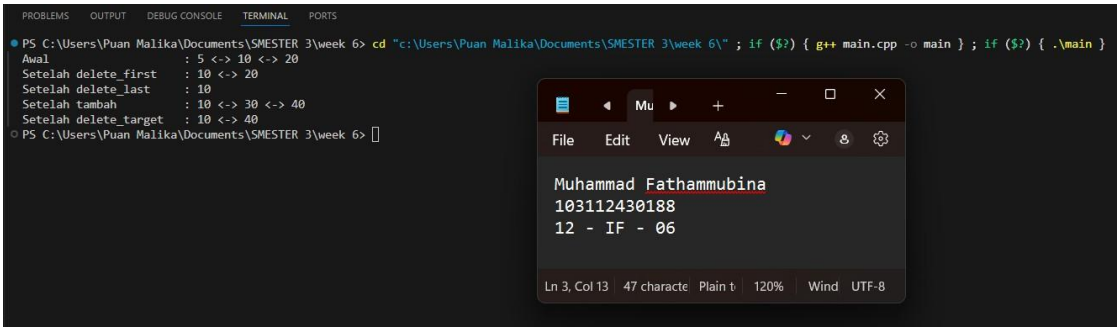
```
Node *ptr_last = NULL;

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){
        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 targetVaue, int newValue){
    Node *current = ptr_first;
    while (current != NULL && current->data != targetVaue)
    {
        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

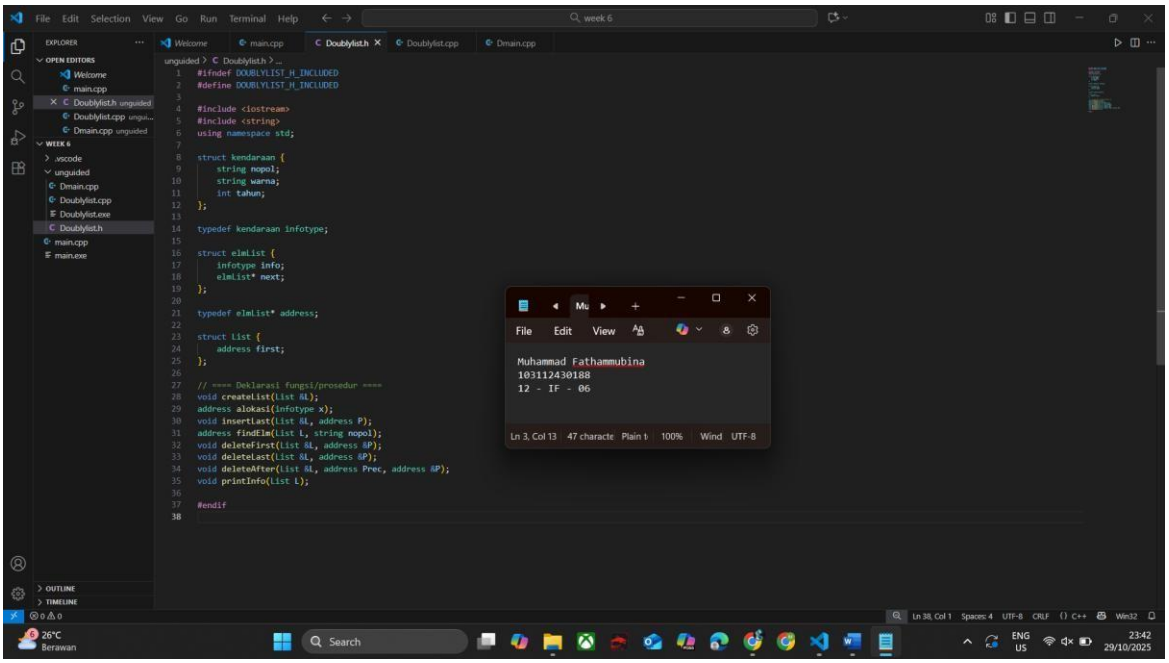


Deskripsi:

Program diatas mengimplementasikan Doubly Linked List Non-Circular menggunakan global pointers (ptr_first dan ptr_last) untuk mengelola daftar data integer. Setiap elemen (Node) dalam daftar memiliki tiga field: data, prev (penunjuk ke node sebelumnya), dan next (penunjuk ke node berikutnya), yang memungkinkan penelusuran dua arah (maju dan mundur). Fungsi add_first dan add_last mengelola penyisipan di ujung-ujung list, menyesuaikan ptr_first atau ptr_last serta menjaga pointer prev dan next agar tetap saling terhubung, sementara add_target mencoba menyisipkan node baru setelah node yang memiliki targetValue. Operasi penghapusan juga lengkap, dengan delete_first, delete_last, dan delete_target yang secara hati-hati memutus dan menyambung kembali pointer prev dan next sebelum membebaskan memori dengan delete. Fungsi view digunakan sebagai traversal maju untuk menampilkan seluruh elemen daftar dengan pemisah <->, menegaskan sifat keterkaitan ganda, dan fungsi edit_node memungkinkan pembaruan data pada node yang ditemukan. Kode ini secara keseluruhan menunjukkan implementasi dasar struktur data Doubly Linked List yang efisien dan aman dalam manajemen memori dan integritas hubungan antar node.

Unguided 1

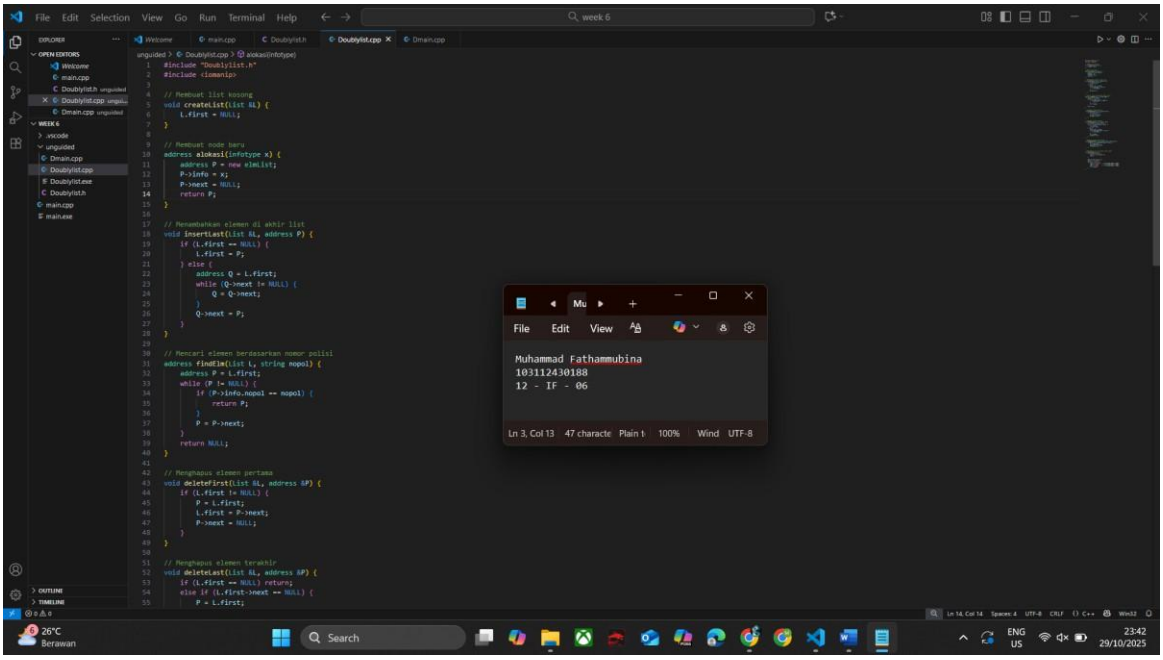
Doublylist.h




```cpp
#ifndef DOUBLYLIST_H_INCLUDED
#define DOUBLYLIST_H_INCLUDED

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

struct kendaraan {
    string nopol;
    string warna;
    int tahun;
};

typedef kendaraan infotype;

struct elmList {
    infotype info;
    elmList* next;
};

typedef elmList* address;

struct List {
    address first;
};

// ==== Deklarasi fungsi/prosedur ====
void createList(List &L);
address alokasi(infotype x);
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(List &L, address Prec, address &P);
void printInfo(List L);

#endif
```

Doublylist.cpp

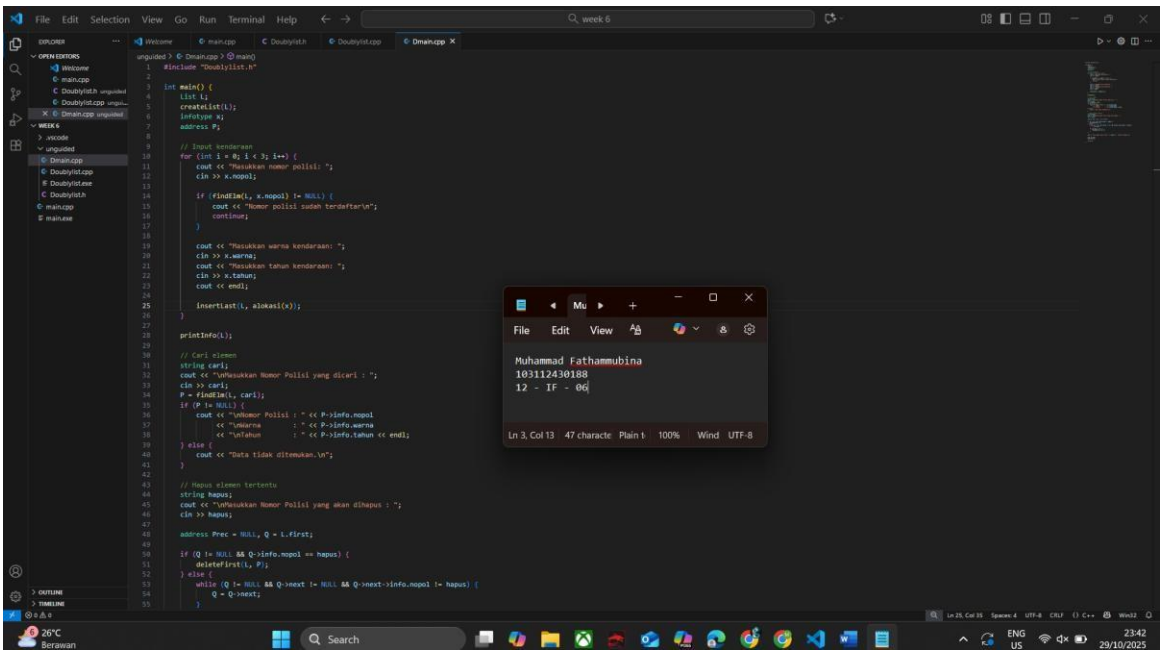



```
#include "Doublylist.h"
#include <iomanip>

// Membuat list kosong
void createList(List &L) {
    L.first = NULL;
}

// Membuat node baru
address alokasi(infotype x) {
    address P = new elmList;
    P->info = x;
    P->next = NULL;
    return P;
}

// Menambahkan elemen di akhir list
void insertLast(List &L, address P) {
    if (L.first == NULL) {
        L.first = P;
```

```
    } else {
        address Q = L.first;
        while (Q->next != NULL) {
            Q = Q->next;
        }
        Q->next = P;
    }
}

// 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;
}

// Menghapus elemen pertama
void deleteFirst(List &L, address &P) {
    if (L.first != NULL) {
        P = L.first;
        L.first = P->next;
        P->next = NULL;
    }
}

// Menghapus elemen terakhir
void deleteLast(List &L, address &P) {
    if (L.first == NULL) return;
    else if (L.first->next == NULL) {
        P = L.first;
        L.first = NULL;
    } else {
        address Q = L.first;
        while (Q->next->next != NULL) {
            Q = Q->next;
        }
        P = Q->next;
        Q->next = NULL;
    }
}

// Menghapus elemen setelah Prec
void deleteAfter(List &L, address Prec, address &P) {
```

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

// Menampilkan isi list sesuai format contoh
void printInfo(List L) {
    if (L.first == NULL) {
        cout << "List kosong.\n";
        return;
    }

    cout << "DATA LIST 1\n";
    address P = L.first;
    while (P != NULL) {
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna    : " << P->info.warna << endl;
        cout << "tahun    : " << P->info.tahun << endl;
        P = P->next;
    }
}
```

Dmain.cpp

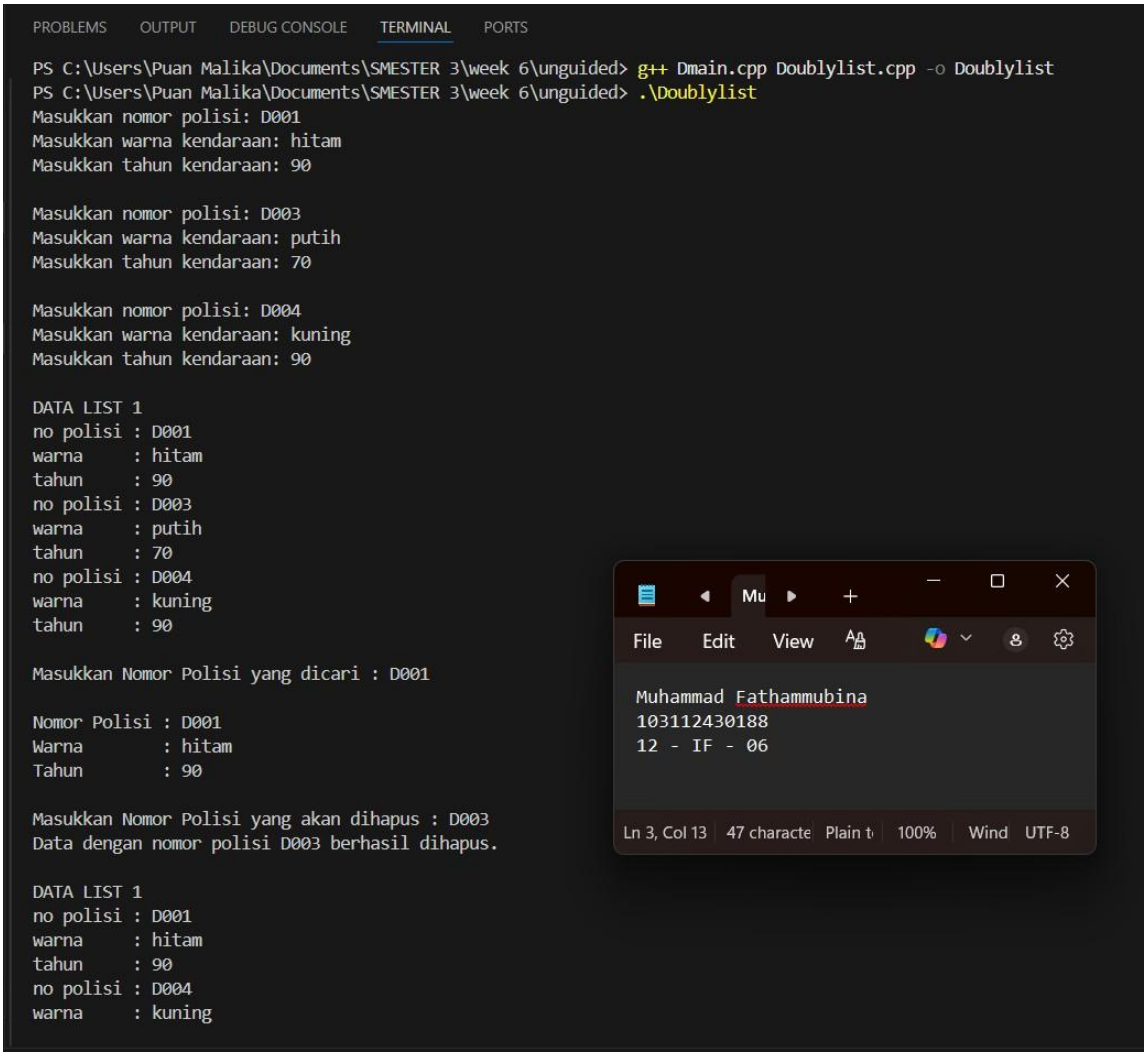



```
#include "Doublylist.h"

int main() {
    List L;
```

```
    createList(L);
    infotype x;
    address P;

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

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

        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.tahun;
        cout << endl;

        insertLast(L, alokasi(x));
    }

    printInfo(L);

    // Cari elemen
    string cari;
    cout << "\nMasukkan Nomor Polisi yang dicari : ";
    cin >> cari;
    P = findElm(L, cari);
    if (P != NULL) {
        cout << "\nNomor Polisi : " << P->info.nopol
             << "\nWarna        : " << P->info.warna
             << "\nTahun        : " << P->info.tahun << endl;
    } else {
        cout << "Data tidak ditemukan.\n";
    }

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

    address Prec = NULL, Q = L.first;

    if (Q != NULL && Q->info.nopol == hapus) {
        deleteFirst(L, P);
    } else {
```

```
        while (Q != NULL && Q->next != NULL && Q->next->info.nopol !=
hapus) {
            Q = Q->next;
        }
        if (Q->next != NULL) {
            deleteAfter(L, Q, P);
        }
    }

    cout << "Data dengan nomor polisi " << hapus << " berhasil
dihapus.\n";

    cout << endl;
    printInfo(L);

    return 0;
}
```

Screenshots Output



Deskripsi:

Program diatas menggunakan ADT untuk mengelola data kendaraan menggunakan Single Linked List. File doublylist.h berisi deklarasi struktur data dan fungsi, sedangkan doublylist.cpp berisi implementasinya, dan main.cpp menjadi program utama untuk interaksi pengguna. Program ini menggunakan #include <iostream> agar dapat memakai fungsi input-output seperti cin dan cout, serta #include <iomanip> untuk membantu pengaturan tampilan teks agar lebih rapi. Variabel Prec dideklarasikan sebagai pointer dengan tipe address, digunakan untuk menyimpan posisi elemen sebelumnya dalam list ketika melakukan operasi penghapusan data, seperti pada prosedur deleteAfter. Program ini memungkinkan pengguna untuk menambah, mencari, menampilkan, dan menghapus data kendaraan berdasarkan nomor polisi dengan format tampilan yang terstruktur dan mudah dibaca.

C. Kesimpulan

Dari hasil analisis praktikum Modul VI tentang program Doubly Linked List, dapat disimpulkan bahwa struktur data ini memberikan fleksibilitas tinggi dalam pengelolaan data secara dinamis. Dengan dua arah penunjuk (prev dan next), proses penyisipan dan penghapusan data dapat dilakukan lebih efisien dibanding singly linked list karena akses ke elemen sebelumnya tidak memerlukan traversal dari awal. Penerapan konsep ADT dalam pemrograman C++ membuat kode lebih terstruktur dan mudah dipelihara. Melalui praktik ini, mahasiswa dapat memahami cara kerja pointer, hubungan antar node, serta logika dasar manipulasi data menggunakan struktur list ganda.

D. Referensi

Muliono, R. (2017). Abstract Data Type (ADT). Universitas Multimedia Nusantara.

Setiyawan, R. D., Hermawan, D., Abdillah, A. F., Mujayanah, A., & Vindua, R. (2023). Penggunaan struktur data stack dalam pemrograman C++ dengan pendekatan array dan linked list. Jurnal Fakultas Ilmu Komputer, Universitas Pamulang.

Wijoyo, A., Prayudi, L. A., Fiqih, M., Santoso, R. D., Putra, R. T. S., Arifin, T., & Farhan, A. (2023). Penggunaan algoritma doubly linked list untuk insertion dan deletion. Fakultas Ilmu Komputer, Universitas Pamulang.