

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

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



Disusun Oleh :

NAMA : DEVI YULIANA

NIM : 103112400151

Dosen

FAHRUDIN MUKTI WIBOWO

STRUKTUR DATA

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Doubly Linked List (DLL) adalah struktur data yang setiap elemennya memiliki dua pointer, yaitu next untuk menunjuk ke elemen sesudahnya dan prev untuk menunjuk ke elemen sebelumnya. Berbeda dengan singly linked list yang hanya bisa diakses satu arah, pada DLL kita bisa menelusuri data baik dari depan maupun dari belakang. Setiap list memiliki pointer first yang menunjuk ke elemen pertama dan last yang menunjuk ke elemen terakhir.

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();
}
```

Screenshots Output

The screenshot shows a terminal window with the following output:

```
PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3> & 'c:\Users\musli\.vscode\extensions\ms-vscode.cpptools-1.28.3-win32-x64\debug
Adapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-450vfj0b.l3a' '--stdout=Microsoft-MIEngine-Out-mxy3vfoq.sw2
' '--stderr=Microsoft-MIEngine-Error-fujukc5t.hki' '--pid=Microsoft-MIEngine-Pid-bpajl5hc.ubh' '--dbgExe=C:\Users\musli\mingw32\b
in\gdb.exe' '--interpreter=mi'
Awal : 5 <-> 10 <-> 20
Setelah delete_first : 10 <-> 20
Setelah delete_last : 10
Setelah tambah : 10 <-> 30 <-> 40
Setelah delete_target : 10 <-> 40
PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3>
```

Overlaid on the terminal is a light blue dialog box with the following text:

```
NAMA : DEVI YULIANA
NIM : 103112400151
KELAS : S1IF-12-06
```

Deskripsi :

Doubly Linked List terdiri dari beberapa node yang saling terhubung dua arah. Setiap node menyimpan nilai (data) serta penunjuk ke node sebelumnya (prev) dan berikutnya (next).

Program ini menggunakan dua pointer utama, yaitu ptr_first untuk node pertama dan ptr_last untuk node terakhir. Beberapa fungsi disediakan untuk mengelola data, seperti add_first() dan add_last() untuk menambah data, delete_first() dan delete_last() untuk menghapus data di awal atau akhir, serta delete_target() dan edit_node() untuk menghapus atau mengubah data tertentu.

Fungsi view() digunakan untuk menampilkan isi list dengan format berantai menggunakan tanda <->. Di bagian main(), program mencoba berbagai operasi seperti menambah, menghapus, dan menampilkan data agar terlihat bagaimana Doubly Linked List bekerja secara dua arah dengan efisien.

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

Unguided 1

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

#define Nil NULL
```

```

// Deklarasi Tipe Data
struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan infotype;

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

struct List {
    ElmList* first;
    ElmList* last;
};

// Deklarasi Fungsi / Prosedur
void createList(List &L);
ElmList* alokasi(infotype x);
void dealokasi(ElmList* &P);
void insertFirst(List &L, ElmList* P);
ElmList* findElm(List L, string nopol);
void printInfo(List L);

// Implementasi
void createList(List &L) {
    L.first = Nil;
    L.last = Nil;
}

ElmList* alokasi(infotype x) {
    ElmList* P = new ElmList;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

```



```

}

void dealokasi(ElmList* &P) {
    delete P;
    P = Nil;
}

// Menambahkan elemen di depan list
void insertFirst(List &L, ElmList* P) {
    if (L.first == Nil) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
    }
}

// Mencari elemen berdasarkan nopol
ElmList* findElm(List L, string nopol) {
    ElmList* P = L.first;
    while (P != Nil) {
        if (P->info.nopol == nopol)
            return P;
        P = P->next;
    }
    return Nil;
}

// Menampilkan seluruh isi list
void printInfo(List L) {
    ElmList* P = L.first;
    if (P == Nil) {
        cout << "List kosong" << endl;
    } else {
        cout << "DATA LIST 1" << endl;
        while (P != Nil) {
            cout << "no polisi : " << P->info.nopol << endl;
            cout << "warna      : " << P->info.warna << endl;
            cout << "tahun       : " << P->info.thnBuat << endl;
        }
    }
}

```

```

        P = P->next;
    }
}

// Program Utama
int main() {
    List L;
    createList(L);
    infotype x;
    char lanjut;

    do {
        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) != Nil) {
            cout << "nomor polisi sudah terdaftar" << endl;
        } else {
            ElmList* P = alokasi(x);
            insertFirst(L, P);
        }

        cout << "Tambah data lagi? (y/t): ";
        cin >> lanjut;
        cout << endl;
    } while (lanjut == 'y' || lanjut == 'Y');

    cout << endl;
    printInfo(L);

    return 0;
}

```

Screenshots Output

```

PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3> cd "c:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\" ; if ($?) { g++
main.cpp -o main } ; if ($?) { .\main }
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90
Tambah data lagi? (y/t): y

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70
Tambah data lagi? (y/t): y

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar
Tambah data lagi? (y/t): y

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90
Tambah data lagi? (y/t): t

DATA LIST 1
no polisi : D004
warna      : kuning
tahun      : 90
no polisi : D003
warna      : putih
tahun      : 70
no polisi : D001
warna      : hitam
tahun      : 90

```

Deskripsi:

Setiap data kendaraan punya tiga atribut, yaitu nomor polisi (nopol), warna, dan tahun pembuatan. Tujuan dari latihan ini adalah supaya bisa memahami gimana cara membentuk list ganda yang bisa diakses dari dua arah (maju dan mundur).

Di program ini, bikin beberapa fungsi dasar seperti createList buat menginisialisasi list kosong, alokasi untuk menyiapkan node baru, dealokasi buat menghapus node dari memori, insertLast atau insertFirst untuk menambahkan data kendaraan, dan printInfo buat menampilkan semua isi list. Hasil akhirnya, program bisa menyimpan beberapa data kendaraan dan menampilkannya dalam urutan tertentu.

Unguided 2

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

#define Nil NULL

// Deklarasi Tipe Data
struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan infotype;

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

struct List {
    ElmList* first;
    ElmList* last;
};

// Deklarasi Fungsi / Prosedur
void createList(List &L);
ElmList* alokasi(infotype x);
void dealokasi(ElmList* &P);
void insertFirst(List &L, ElmList* P);
ElmList* findElm(List L, string nopol);
void printInfo(List L);
void cariKendaraan(List L);

// Implementasi
void createList(List &L) {
    L.first = Nil;
    L.last = Nil;
}
```

```

ElmList* alokasi(infotype x) {
    ElmList* P = new ElmList;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

void dealokasi(ElmList* &P) {
    delete P;
    P = Nil;
}

void insertFirst(List &L, ElmList* P) {
    if (L.first == Nil) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
    }
}

ElmList* findElm(List L, string nopol) {
    ElmList* P = L.first;
    while (P != Nil) {
        if (P->info.nopol == nopol)
            return P;
        P = P->next;
    }
    return Nil;
}

void printInfo(List L) {
    ElmList* P = L.first;
    if (P == Nil) {
        cout << "List kosong" << endl;
    } else {
        cout << "DATA LIST 1" << endl;
    }
}

```

```

        while (P != Nil) {
            cout << "no polisi : " << P->info.nopol << endl;
            cout << "warna      : " << P->info.warna << endl;
            cout << "tahun      : " << P->info.thnBuat << endl;
            P = P->next;
        }
    }
}

// Fungsi tambahan untuk latihan 2 (menampilkan hasil
pencarian)
void cariKendaraan(List L) {
    string cari;
    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> cari;

    ElmList* P = findElm(L, cari);
    if (P != Nil) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna      : " << P->info.warna << endl;
        cout << "Tahun      : " << P->info.thnBuat << endl;
    } else {
        cout << "Data dengan nomor polisi " << cari << " tidak
ditemukan." << endl;
    }
}

// Program Utama
int main() {
    List L;
    createList(L);
    infotype x;
    char lanjut;

    // Input data kendaraan
    do {
        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) != Nil) {
            cout << "nomor polisi sudah terdaftar" << endl;
        } else {
            ElmList* P = alokasi(x);
            insertFirst(L, P);
        }

        cout << "Tambah data lagi? (y/t): ";
        cin >> lanjut;
        cout << endl;
    } while (lanjut == 'y' || lanjut == 'Y');

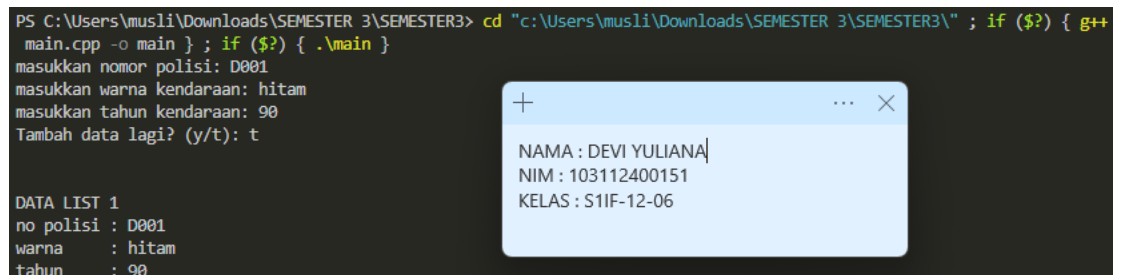
    cout << endl;
    printInfo(L);
    cout << endl;

    // Pencarian data kendaraan
    cariKendaraan(L);

    return 0;
}

```

Screenshots Output



```

PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3> cd "c:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\" ; if ($?) { g++
main.cpp -o main } ; if ($?) { .\main }
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90
Tambah data lagi? (y/t): t

DATA LIST 1
no polisi : D001
warna     : hitam
tahun     : 90

```

Deskripsi:

Di bagian ini, menambahkan fitur supaya pengguna bisa mencari kendaraan berdasarkan nomor polisi. Jadi, ketika pengguna mengetik nomor polisi tertentu, program akan menelusuri list dari awal sampai akhir untuk mencari datanya. Kalau ketemu, program akan menampilkan detail kendaraan seperti nomor polisi, warna, dan tahun buatnya.

Unguided 3

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

#define Nil NULL

// Deklarasi Tipe Data
struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan infotype;

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

struct List {
    ElmList* first;
    ElmList* last;
};

// Deklarasi Fungsi / Prosedur
void createList(List &L);
ElmList* alokasi(infotype x);
void dealokasi(ElmList* &P);
void insertFirst(List &L, ElmList* P);
ElmList* findElm(List L, string nopol);
void printInfo(List L);
void cariKendaraan(List L);

// delete operations
void deleteFirst(List &L, ElmList* &P);
void deleteLast(List &L, ElmList* &P);
void deleteAfter(ElmList* Prec, ElmList* &P);
void deleteByNopol(List &L, string nopol);
```



```

// Implementasi
void createList(List &L) {
    L.first = Nil;
    L.last = Nil;
}

ElmList* alokasi(infotype x) {
    ElmList* P = new ElmList;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

void dealokasi(ElmList* &P) {
    delete P;
    P = Nil;
}

void insertFirst(List &L, ElmList* P) {
    if (L.first == Nil) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
    }
}

ElmList* findElm(List L, string nopol) {
    ElmList* P = L.first;
    while (P != Nil) {
        if (P->info.nopol == nopol)
            return P;
        P = P->next;
    }
    return Nil;
}

```

```

void printInfo(List L) {
    ElmList* P = L.first;
    if (P == Nil) {
        cout << "List kosong" << endl;
    } else {
        cout << "DATA LIST 1" << endl;
        while (P != Nil) {
            cout << "no polisi : " << P->info.nopol << endl;
            cout << "warna      : " << P->info.warna << endl;
            cout << "tahun      : " << P->info.thnBuat << endl;
            P = P->next;
        }
    }
}

void cariKendaraan(List L) {
    string cari;
    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> cari;

    ElmList* P = findElm(L, cari);
    if (P != Nil) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna          : " << P->info.warna << endl;
        cout << "Tahun           : " << P->info.thnBuat << endl;
    } else {
        cout << "Data dengan nomor polisi " << cari << " tidak
ditemukan." << endl;
    }
}

// DELETE OPERATIONS

// Hapus elemen pertama
void deleteFirst(List &L, ElmList* &P) {
    if (L.first == Nil) {
        P = Nil;
    } else if (L.first == L.last) {
        P = L.first;
        L.first = Nil;
        L.last = Nil;
    }
}

```

```

    } else {
        P = L.first;
        L.first = L.first->next;
        L.first->prev = Nil;
        P->next = Nil;
    }
}

// Hapus elemen terakhir
void deleteLast(List &L, ElmList* &P) {
    if (L.first == Nil) {
        P = Nil;
    } else if (L.first == L.last) {
        P = L.last;
        L.first = Nil;
        L.last = Nil;
    } else {
        P = L.last;
        L.last = L.last->prev;
        L.last->next = Nil;
        P->prev = Nil;
    }
}

// Hapus elemen setelah Prec
void deleteAfter(ElmList* Prec, ElmList* &P) {
    if (Prec != Nil && Prec->next != Nil) {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != Nil) {
            P->next->prev = Prec;
        }
        P->next = Nil;
        P->prev = Nil;
    }
}

// Hapus elemen berdasarkan nopol
void deleteByNopol(List &L, string nopol) {
    ElmList* P = findElm(L, nopol);
    if (P == Nil) {

```

```

        cout << "Data tidak ditemukan." << endl;
        return;
    }

    if (P == L.first) {
        deleteFirst(L, P);
    } else if (P == L.last) {
        deleteLast(L, P);
    } else {
        deleteAfter(P->prev, P);
    }

    cout << "Data dengan nopol " << nopol << " berhasil
dihapus." << endl;
    dealokasi(P);
}

// Program Utama
int main() {
    List L;
    createList(L);
    infotype x;
    char lanjut;

    // Input data kendaraan
    do {
        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) != Nil) {
            cout << "nomor polisi sudah terdaftar" << endl;
        } else {
            ElmList* P = alokasi(x);
            insertFirst(L, P);
        }

        cout << "Tambah data lagi? (y/t): ";
    } while (lanjut == 'y');
}

```

```

        cin >> lanjut;
        cout << endl;
    } while (lanjut == 'y' || lanjut == 'Y');

    cout << endl;
    printInfo(L);
    cout << endl;

    // Cari kendaraan
    cariKendaraan(L);
    cout << endl;

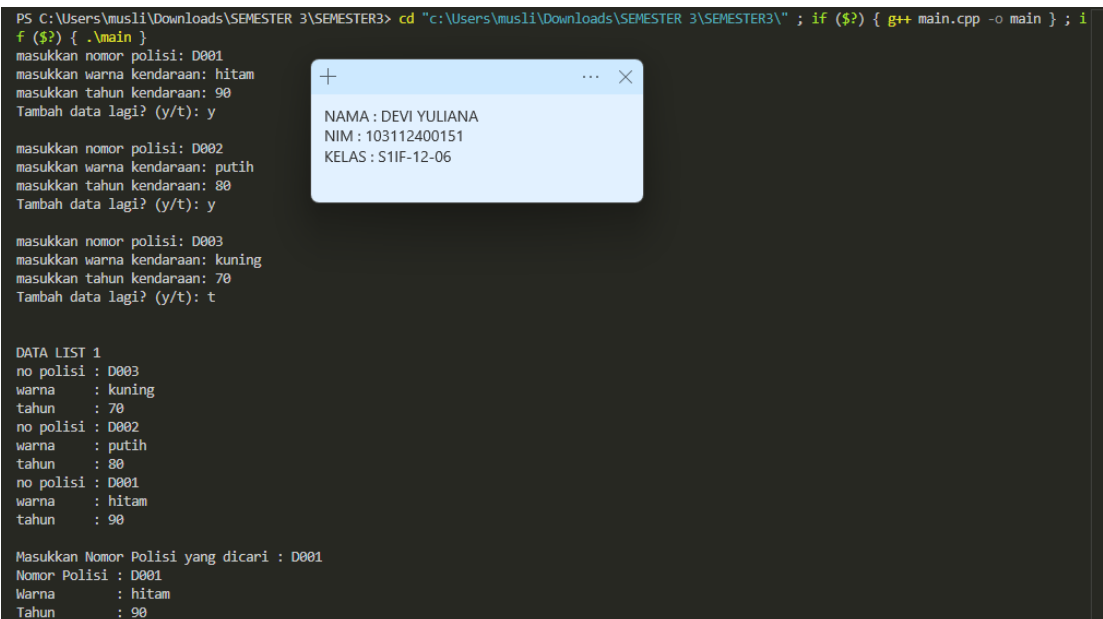
    // Hapus kendaraan
    string hapus;
    cout << "Masukkan nomor polisi yang ingin dihapus: ";
    cin >> hapus;
    deleteByNopol(L, hapus);

    cout << endl;
    printInfo(L);

    return 0;
}

```

Screenshots Output



```

PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3> cd "c:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\" ; if ($?) { g++ main.cpp -o main } ; i
f ($?) { .\main }
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90
Tambah data lagi? (y/t): y

masukkan nomor polisi: D002
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 80
Tambah data lagi? (y/t): y

masukkan nomor polisi: D003
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 70
Tambah data lagi? (y/t): t

DATA LIST 1
no polisi : D003
warna      : kuning
tahun      : 70
no polisi : D002
warna      : putih
tahun      : 80
no polisi : D001
warna      : hitam
tahun      : 90

Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi : D001
Warna         : hitam
Tahun         : 90

```

```
Masukkan Nomor Polisi yang dicari : D001
Nomor Polisi : D001
Warna      : hitam
Tahun      : 90

Masukkan nomor polisi yang ingin dihapus: D003
Data dengan nopol D003 berhasil dihapus.

DATA LIST 1
no polisi : D002
warna     : putih
tahun     : 80
no polisi : D001
warna     : hitam
tahun     : 90
```

Deskripsi:

Dibagian ini, diminta untuk membuat tiga prosedur penghapusan: deleteFirst, deleteLast, dan deleteAfter. Ketiga fungsi ini punya tugas berbeda tergantung posisi elemen yang mau dihapus, apakah di awal, di akhir, atau di tengah list. Selain itu, juga tambahkan prosedur deleteByNopol, biar pengguna bisa hapus data kendaraan langsung berdasarkan nomor polisi.

Dengan ini, bisa ngerti gimana cara mengatur ulang pointer next dan prev biar list tetap nyambung setelah elemen dihapus. Di akhir program, pengguna bisa lihat hasil list setelah data dihapus, dan program tetap berjalan dengan baik.

D. Kesimpulan

Doubly Linked List adalah struktur data yang memungkinkan akses data dua arah karena setiap elemen memiliki pointer next dan prev. Operasi seperti menambah, mencari, dan menghapus data memerlukan pemahaman yang baik tentang cara kerja pointer agar hubungan antar-elemen tetap terjaga. Selain itu, pengelolaan memori juga penting supaya tidak terjadi error atau kebocoran data.

E. Referensi

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++ (Vol. 1)*. CV. AA. RIZKY.