**LAPORAN PRAKTIKUM**
**STRUKTUR DATA**

**MODUL 6**
**DOUBLY LINKED LIST**

**Disusun Oleh :**
NAMA : LOH SUWARGI NITIS HAMENGKU BINTANG
NIM : 103112400116

**Dosen**
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY PURWOKERTO**
**2025**

A. Dasar Teori

Struktur data Doubly Linked List (DLL) adalah sebuah daftar berantai di mana setiap node menyimpan setidaknya tiga komponen utama: data, pointer ke node berikutnya (next), dan pointer ke node sebelumnya (prev). Karena adanya pointer dua arah ini, traversal dapat dilakukan dari awal ke akhir maupun sebaliknya, yang memberikan fleksibilitas lebih dibanding singly linked list. Studi oleh Penggunaan Algoritma Doubly Linked List Untuk Insertion Dan Deletion (Wijoyo et al., 2024) menunjukkan bahwa implementasi DLL terbukti efisien dalam operasi penyisipan dan penghapusan data secara dinamis—karena pointer prev memungkinkan penghapusan suatu node tanpa perlu traversal kembali dari head ke posisi sebelumnya. Dalam konteks aplikasi nyata, hal ini menjadikan DLL ideal untuk sistem yang sering melakukan penambahan atau pengurangan elemen secara acak atau terus-menerus, seperti undo/redo pada aplikasi atau history navigasi.

Operasi utama pada DLL mencakup penyisipan (insert) dan penghapusan (delete) di berbagai posisi—awal, akhir, atau tengah—dan traversal dua arah. Penelitian terbaru menegaskan bahwa melalui DLL, penghapusan node yang sudah diketahui posisinya dapat dilakukan dalam waktu konstan (O(1)) karena pointer prev langsung menunjuk ke node sebelumnya, tanpa perlu pencarian ulang. Sebagai contoh, bagaimana pointer next dan prev diperbaharui saat penghapusan dijelaskan secara rinci dalam literatur (misalnya di artikel deletion pada DLL). Operasi-operasi ini menjadikan DLL sangat sesuai untuk aplikasi yang membutuhkan manipulasi daftar secara fleksibel dan dinamis.

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 ? " <->" : "");
```

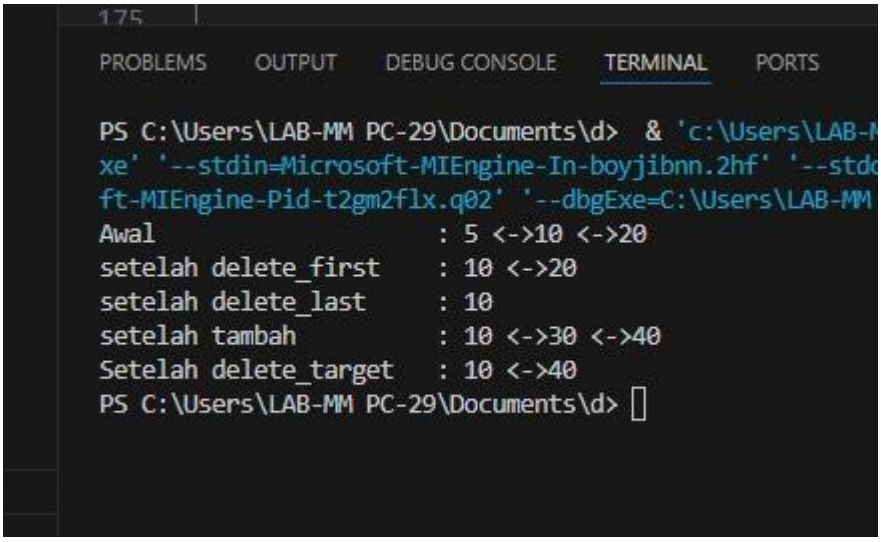
```
        current = current->next;
    }
    cout << endl;
}

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;
        }
        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




Deskripsi: Program ini mendemonstrasikan cara kerja struktur data Doubly Linked List (DLL), yaitu daftar berantai ganda yang memungkinkan pergerakan dua arah antar elemen. Masing-masing elemen atau node terdiri atas tiga bagian: data, pointer ke node sebelumnya (prev), dan pointer ke node berikutnya (next). Dua pointer utama, ptr_first dan ptr_last, digunakan untuk mengontrol node pertama dan terakhir dalam daftar.

Melalui berbagai fungsi yang tersedia, pengguna dapat melakukan operasi penyisipan seperti menambah data di awal, di akhir, atau setelah data tertentu. Fungsi penghapusan juga disediakan untuk menghapus node pertama, terakhir, atau node dengan nilai tertentu. Terdapat pula fungsi edit_node untuk memperbarui isi node dan view untuk menampilkan seluruh elemen. Dengan struktur modular dan penggunaan pointer ganda, program ini memberikan ilustrasi jelas tentang cara pengelolaan memori dinamis dalam linked list yang bersifat dua arah.

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

Unguided 1

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

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

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

typedef kendaraan infotype;

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

typedef ElmList* address;

struct List {
   address first;
   address last;
};

void createList(List &L);
address alokasi(infotype x);
```

```
void dealokasi(address &P);
void insertFirst(List &L, address P);
void insertLast(List &L, address P);
address searchByNopol(List L, string nopol);
void printInfo(List L);

#endif
```

```
#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;
}
```

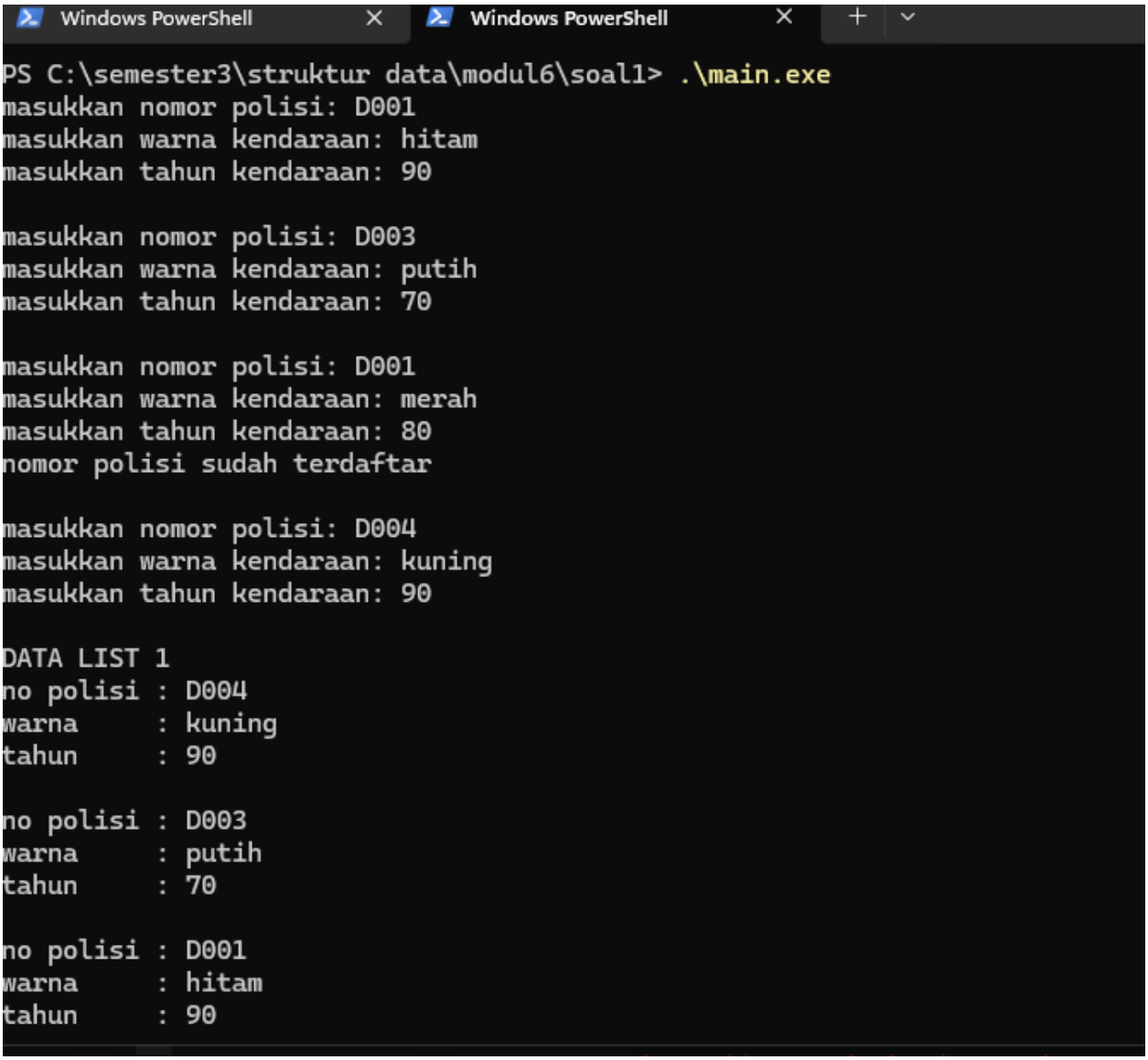
```
void insertFirst(List &L, address P) {
   if (L.first == NULL) {
      L.first = P;
      L.last = P;
   } else {
      P->next = L.first;
      L.first->prev = P;
      L.first = P;
   }
}

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

```
    return NULL;
}

void printInfo(List L) {
    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;
        cout << endl;
        P = P->next;
    }
}
```

```
#include "Doublylist.h"
#include <iostream>
using namespace std;

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

    infotype x;

    for (int i = 0; i < 4; i++) {
        cout << "masukkan nomor polisi: ";
        cin >> x.nopol;
        cout << "masukkan warna kendaraan: ";
```

```
        cin >> x.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> x.tahun;

        address P = searchByNopol(L, x.nopol);
        if (P != NULL) {
            cout << "nomor polisi sudah terdaftar" << endl;
        } else {
            insertFirst(L, alokasi(x));
        }
        cout << endl;
    }

    cout << "DATA LIST 1" << endl;
    printInfo(L);

    return 0;
}
```

Screenshots Output




Deskripsi: Program ini membuat struktur data Doubl linked list untuk menyimpan data kendaraan. Setiap data kendaraan memiliki atribut no polisi, warna, dan tahun. Struktur ini memungkinkan penyimpanan data secara dinamis tanpa batasan jumlah elemen.

Selain itu, program juga memastikan tidak ada data dengan nomor kendaraan yang sama melalui proses pengecekan duplikat. Jika pengguna mencoba memasukkan nomor yang sudah ada, sistem akan menolak input tersebut. Hasil akhirnya, program akan menampilkan seluruh data kendaraan yang berhasil tersimpan di dalam list menggunakan fungsi printInfo.

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

Unguided 2

```
#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;
   ElmList *prev;
};

typedef ElmList* address;

struct List {
   address first;
   address last;
};

void createList(List &L);
address alokasi(infotype x);
void insertFirst(List &L, address P);
void printInfo(List L);
```

```
address searchByNopol(List L, string nopol);
address findElm(List L, string nopol);

#endif
```

```
#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 insertFirst(List &L, address P) {
    if (L.first == NULL) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
```

```
    }
}

void printInfo(List L) {
    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;
    }
}

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

address findElm(List L, string nopol) {
    address P = L.first;
    while (P != NULL) {
        if (P->info.nopol == nopol) {
            cout << "Nomor Polisi  : " << P->info.nopol << endl;
            cout << "Warna         : " << P->info.warna << endl;
```

```
            cout << "Tahun         : " << P->info.tahun << endl;
            return P;
        }
        P = P->next;
    }
    cout << "Data dengan nomor polisi " << nopol << " tidak ditemukan." << endl;
    return NULL;
}
```

```
#include "Doublylist.h"
#include <iostream>
using namespace std;

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

    infotype x;

    for (int i = 0; i < 4; i++) {
        cout << "Masukkan nomor polisi: ";
        cin >> x.nopol;
        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.tahun;

        address P = searchByNopol(L, x.nopol);
```

```
            if (P != NULL) {
                cout << "Nomor polisi sudah terdaftar!" << endl;
            } else {
                insertFirst(L, alokasi(x));
            }
            cout << endl;
        }

        cout << "\nDATA LIST 1" << endl;
        printInfo(L);

        cout << "\nMasukkan Nomor Polisi yang ingin dicari: ";
        string cari;
        cin >> cari;
        findElm(L, cari);

        return 0;
    }
```

Screenshots Output

```
Masukkan nomor polisi: D003
Masukkan warna kendaraan: putih
Masukkan tahun kendaraan: 70

Masukkan nomor polisi: D001
Masukkan warna kendaraan: merah
Masukkan tahun kendaraan: 90
Nomor polisi sudah terdaftar!

Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90
Nomor polisi sudah terdaftar!


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

Masukkan Nomor Polisi yang ingin dicari: D001
Nomor Polisi  : D001
Warna         : hitam
Tahun         : 90
PS C:\semester3\struktur data\modul6\soal2>
```

Deskripsi: Program ini dikembangkan dengan menambahkan kemampuan insert dan traversal pada Doubly Linked List. Fungsi insertLast digunakan untuk menambahkan data kendaraan di bagian akhir list, sehingga data tersusun sesuai urutan input pengguna. Fungsi printInfo digunakan untuk menampilkan seluruh data kendaraan yang tersimpan di dalam list secara berurutan.

Melalui program ini, pengguna dapat memahami bagaimana setiap node baru disambungkan dengan node sebelumnya menggunakan pointer prev dan next. Selain itu, program juga menunjukkan bagaimana list dapat ditelusuri dari awal hingga akhir untuk menampilkan seluruh elemen dengan rapi.

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

Unguided 3

```
#ifndef DOUBLYLIST_H
```

```
#define DOUBLYLIST_H

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

struct kendaraan {
   string nomor;
   string warna;
   int tahun;
};
typedef kendaraan infotype;

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

struct List {
   address first;
   address last;
};

void createList(List &L);
address alokasi(infotype x);
void insertLast(List &L, address P);
void printInfo(List L);
```

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

#endif
```

```
#include "Doublylist.h"
#include <iostream>
using namespace std;

void createList(List &L) {
    L.first = nullptr;
    L.last = nullptr;
}

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

void insertLast(List &L, address P) {
    if (L.first == nullptr) {
        L.first = P;
        L.last = P;
    } else {
```

```
        L.last->next = P;
        P->prev = L.last;
        L.last = P;
    }
}

void printInfo(List L) {
    address P = L.first;
    if (P == nullptr) {
        cout << "List kosong." << endl;
        return;
    }
    while (P != nullptr) {
        cout << "Nomor : " << P->info.nomor << endl;
        cout << "Warna : " << P->info.warna << endl;
        cout << "Tahun : " << P->info.tahun << endl;
        cout << endl;
        P = P->next;
    }
}

void deleteFirst(List &L, address &P) {
    if (L.first == nullptr) {
        P = nullptr;
        return;
    }

    P = L.first;
    if (L.first == L.last) {
        L.first = nullptr;
```

```
        L.last = nullptr;
    } else {
        L.first = P->next;
        L.first->prev = nullptr;
        P->next = nullptr;
    }
}

void deleteLast(List &L, address &P) {
    if (L.first == nullptr) {
        P = nullptr;
        return;
    }

    P = L.last;
    if (L.first == L.last) {
        L.first = nullptr;
        L.last = nullptr;
    } else {
        L.last = P->prev;
        L.last->next = nullptr;
        P->prev = nullptr;
    }
}

void deleteAfter(List &L, address Prec, address &P) {
    if (Prec == nullptr || Prec->next == nullptr) {
        P = nullptr;
        return;
    }
```

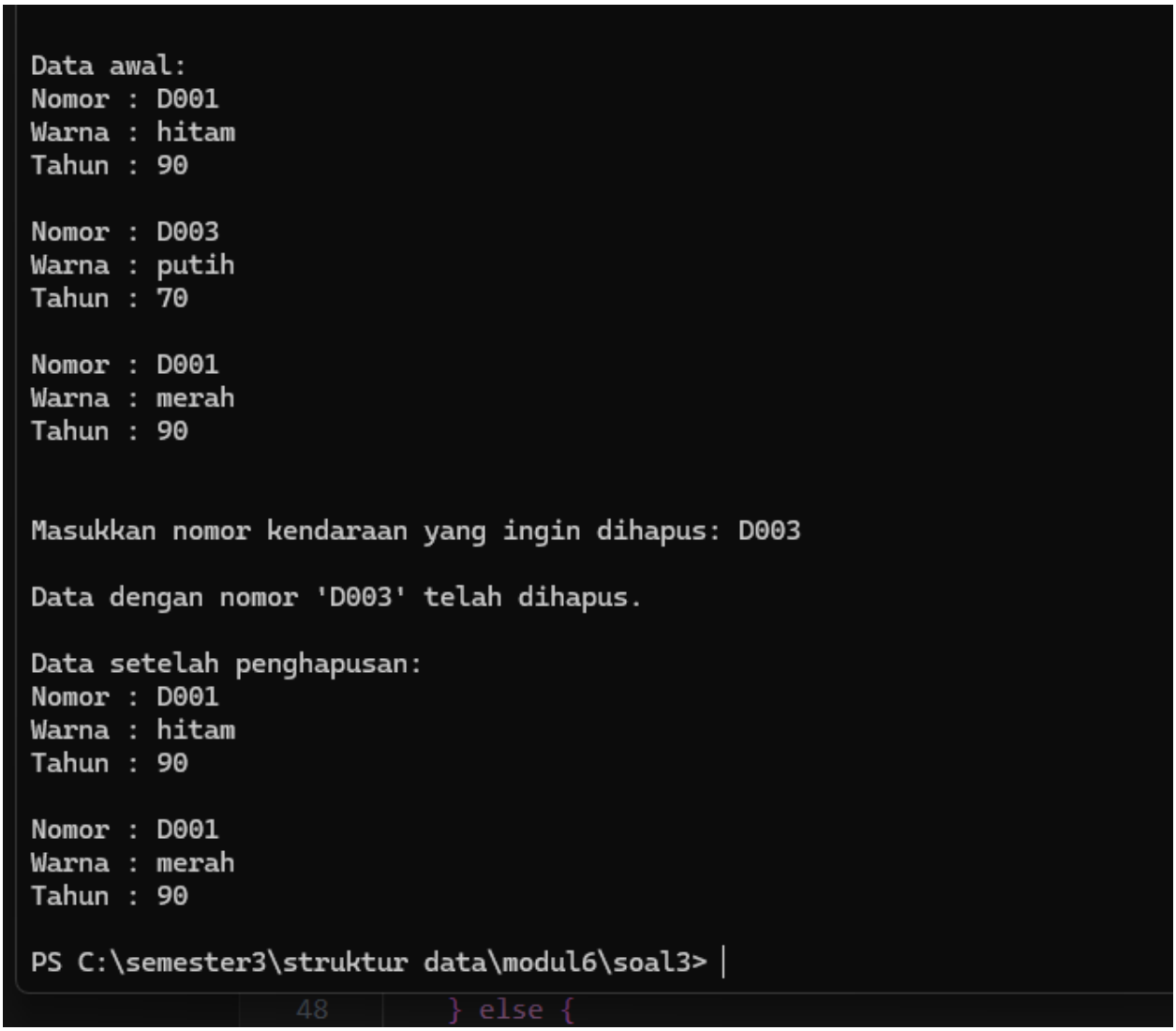
```
    P = Prec->next;
    Prec->next = P->next;
    if (P->next != nullptr) {
        P->next->prev = Prec;
    } else {
        L.last = Prec;
    }
    P->next = nullptr;
    P->prev = nullptr;
}
```

```
#include "Doublylist.h"
#include <iostream>
using namespace std;

address searchByNomor(List L, string nomor) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info.nomor == nomor) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

int main() {
```

```
List L;
createList(L);
infotype x;

for (int i = 0; i < 3; i++) {
    cout << "Masukkan nomor: ";
    cin >> x.nomor;
    cout << "Masukkan warna: ";
    cin >> x.warna;
    cout << "Masukkan tahun: ";
    cin >> x.tahun;

    insertLast(L, alokasi(x));
    cout << endl;
}

cout << "\nData awal:\n";
printInfo(L);

string hapusNomor;
cout << "\nMasukkan nomor kendaraan yang ingin dihapus: ";
cin >> hapusNomor;

address target = searchByNomor(L, hapusNomor);
address P = nullptr;

if (target == nullptr) {
    cout << "\nData dengan nomor '" << hapusNomor << "' tidak ditemukan.\n";
} else {
    if (target == L.first) {
        deleteFirst(L, P);
    }
```

```
        else if (target == L.last) {
            deleteLast(L, P);
        }
        else {
            address prec = target->prev;
            deleteAfter(L, prec, P);
        }

        cout << "\nData dengan nomor '" << hapusNomor << "' telah dihapus.\n";
        delete P;
    }

    cout << "\nData setelah penghapusan:\n";
    printInfo(L);

    return 0;
}
```

Screenshots Output




Deskripsi Program ini memberikan opsi kepada pengguna untuk menghapus data tertentu dengan mencari nomor kendaraan terlebih dahulu, kemudian menghapus node tersebut menggunakan fungsi delete yang sesuai. Proses penghapusan harus memperhatikan pengaturan pointer prev dan next agar struktur list tetap konsisten.

Selain itu, versi pengembangan program juga menambahkan fitur agar pengguna dapat memilih data yang ingin dihapus berdasarkan nomor kendaraan. Dengan cara ini, pengguna lebih fleksibel dalam menentukan data mana yang ingin dihapus, sehingga simulasi Doubly Linked List terasa lebih nyata dan interaktif.

## G. Kesimpulan

Program Doubly Linked List yang telah dibuat menunjukkan bagaimana data dapat disusun secara dinamis dan fleksibel melalui penggunaan pointer ganda. Setiap node memiliki keterkaitan dua arah yang memungkinkan operasi penelusuran, penyisipan, dan penghapusan dilakukan dengan efisien tanpa perlu memindahkan seluruh data. Hal ini membuktikan bahwa struktur data linked list lebih unggul dalam pengelolaan data berukuran besar atau yang sering mengalami perubahan dibandingkan struktur data statis seperti array.

Dengan memanfaatkan fungsi-fungsi yang modular, program ini juga melatih pemahaman tentang manajemen memori dan hubungan antar-node dalam pemrograman berbasis pointer. Konsep ini menjadi pondasi penting dalam pengembangan algoritma dan struktur data tingkat lanjut.

## H. Referensi

**Lohmann, S., & Tutsch, D. (2023). The Doubly Linked Tree of Singly Linked Rings: Providing Hard Real-Time Database Operations on an FPGA. *Computers*, *13*(1), 8.**

**Erkamim, E., Abdurrohim, I., Yuliyanti, S., Karim, R., Rahman, A., Admira, T. M. A., & Ridwan, A. (2024). *Buku Ajar Algoritma dan Struktur Data*. PT. Sonpedia Publishing Indonesia.**