LAPORAN PRAKTIKUM
Modul VI
“Double Linked List Bagian Pertama”

Disusun Oleh:
Dewi Atika Muthi -2211104042
SE-07-02

Dosen:
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

Tujuan praktikum ini adalah:

- Memahami konsep modul linked list
- Mengaplikasikan konsep double linked list dengan menggunakan pointer dan bahasa C

2. Landasan Teori

Double Linked List adalah struktur data linear yang memiliki karakteristik unik:

- Setiap elemen memiliki dua buah pointer:
  1. Pointer prev yang menunjuk ke elemen sebelumnya
  2. Pointer next yang menunjuk ke elemen selanjutnya

- Komponen utama dalam Double Linked List:
  1. First: Pointer yang menunjuk elemen pertama list
  2. Last: Pointer yang menunjuk elemen terakhir list
  3. Next: Pointer yang menunjuk elemen di depannya
  4. Prev: Pointer yang menunjuk elemen di belakangnya

- Operasi dasar pada Double Linked List meliputi:
  1. Insert (Penyisipan):
     - Insert First (Menyisipkan di awal)

```
void insertFirst(list &L, address &P) {
    // Jika list kosong
    if (first(L) == NULL) {
        first(L) = P;
        last(L) = P;
    } else {
        // Sambungkan node baru ke depan list
        next(P) = first(L);
        prev(first(L)) = P;
        first(L) = P;
    }
}
```

     - Insert Last (Menyisipkan di akhir)

```
void insertLast(list &L, address &P) {
    // Jika list kosong
    if (first(L) == NULL) {
        first(L) = P;
        last(L) = P;
    } else {
        // Sambungkan node baru di akhir list
        prev(P) = last(L);
        next(last(L)) = P;
        last(L) = P;
    }
}
```

     - Insert After (Menyisipkan setelah elemen tertentu)

```
void insertAfter(address Prec, address P) {
    // Sambungkan node baru setelah node Prec
    next(P) = next(Prec);
    prev(P) = Prec;
    prev(next(Prec)) = P;
    next(Prec) = P;
}
```

- Insert Before (Menyisipkan sebelum elemen tertentu)

```
void insertBefore(address Prec, address P) {
    // Sambungkan node baru sebelum node Prec
    prev(P) = prev(Prec);
    next(P) = Prec;
    next(prev(Prec)) = P;
    prev(Prec) = P;
}
```

2. Delete (Penghapusan):
   - Delete First (Menghapus elemen pertama)

```
void deleteFirst(list &L, address &P) {
    P = first(L);
    first(L) = next(first(L));

    // Jika list tinggal satu elemen
    if (first(L) == NULL) {
        last(L) = NULL;
    } else {
        prev(first(L)) = NULL;
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

   - Delete Last (Menghapus elemen terakhir)

```
void deleteLast(list &L, address &P) {
    P = last(L);
    last(L) = prev(last(L));

    // Jika list tinggal satu elemen
    if (last(L) == NULL) {
        first(L) = NULL;
    } else {
        next(last(L)) = NULL;
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

   - Delete After (Menghapus setelah elemen tertentu)

```
void deleteAfter(address Prec, address &P) {
    P = next(Prec);
    next(Prec) = next(P);

    if (next(P) != NULL) {
        prev(next(P)) = Prec;
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

   - Delete Before (Menghapus sebelum elemen tertentu)

```
void deleteBefore(address Prec, address &P) {
    P = prev(Prec);
    prev(Prec) = prev(P);
```

```
    if (prev(P) != NULL) {
        next(prev(P)) = Prec;
    }

    // Putuskan node yang dihapus
    next(P) = NULL;
    prev(P) = NULL;
}
```

3. Searching, Update, dan View:
   - Searching

```
address findElm(list L, infotype x) {
    address P = first(L);
    while (P != NULL) {
        if (info(P) == x) {
            return P;
        }
        P = next(P);
    }
    return NULL;
}}
```

   - Update

```
void updateElm(address &P, infotype newInfo) {
    info(P) = newInfo;
}
```

   - View/Print:

```
void printInfo(list L) {
    address P = first(L);
    while (P != NULL) {
        // Cetak informasi elemen
        printf("%s\n", info(P));
        P = next(P);
    }
}
```

4. Guided

Implementasi Double Linked List pada C++

1) Definisi Kelas Node

```
class Node {
public:
    int data;
    Node* prev;
    Node* next;
};
```

Penjelasan:

- `data`: Menyimpan nilai data yang disimpan pada node.
- `prev`: Pointer ke node sebelumnya dalam Double Linked List.
- `next`: Pointer ke node selanjutnya dalam Double Linked List.

2) Definisi Kelas DoubleyLinkedList

```
class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
```

```
    // Constructor untuk inisialisasi head dan tail
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }
```

Penjelasan:

- `head`: Pointer ke node pertama dalam Double Linked List.
- `tail`: Pointer ke node terakhir dalam Double Linked List.

3) Operasi Penambahan Elemen

```
// Fungsi untuk menambahkan elemen di depan list
    void insert(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;
        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode; // Jika list kosong, tail juga mengarah ke
   node baru
        }
        head = newNode;
    }
```

Penjelasan:

Fungsi `insert` akan menambahkan sebuah node baru dengan data data di awal Double Linked List.

- Alokasi memori untuk node baru menggunakan `newNode`.
- Isi field data pada node baru dengan nilai data.
- Set `prev` pointer pada node baru menjadi `nullptr`, karena node baru akan menjadi `head`.
- Set `next` pointer pada node baru menjadi `head`, sehingga node baru akan menjadi `head` baru.
- Jika `head` tidak `nullptr` (list tidak kosong), set `prev` pointer pada node lama (`head` lama) agar menunjuk ke node baru.
- Jika `head nullptr` (list kosong), set `tail` agar menunjuk ke node baru, karena node baru menjadi satu-satunya elemen.
- Pindahkan `head` agar menunjuk ke node baru.

4) Operasi Penghapusan Elemen

```
// Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
```

```
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }
```

Penjelasan:

Fungsi `deleteNode` akan menghapus node pertama dari Double Linked List.

- Jika `head` adalah `nullptr` (list kosong), fungsi langsung kembali.
- Simpan pointer ke node pertama (`head`) pada variabel sementara `temp`.
- Pindahkan `head` agar menunjuk ke node setelah node pertama.
- Jika `head` tidak `nullptr` (list masih berisi elemen), set `prev` pointer pada node baru `head` menjadi `nullptr`.
- Jika `head nullptr` (list hanya berisi satu elemen), set `tail` menjadi `nullptr j`uga.
- Hapus node pertama yang disimpan pada `temp`.

5) Operasi Update Elemen

```
// Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }
```

Penjelasan:

Fungsi update akan mengubah nilai data pada node yang memiliki nilai oldData menjadi newData.

- Mulai dari node pertama (`head`), iterasi melalui seluruh list menggunakan pointer `current`.
- Pada setiap `node`, periksa apakah data pada node tersebut sama dengan `oldData`.
- Jika ditemukan, ubah nilai data pada node tersebut menjadi `newData` dan kembalikan `true` sebagai tanda data berhasil diperbarui.
- Jika tidak ditemukan, lanjutkan iterasi ke node selanjutnya.
- Jika penelusuran seluruh list selesai tanpa menemukan `oldData`, kembalikan `false` sebagai tanda data tidak ditemukan.

6) Operasi Penghapusan Seluruh Elemen

```
// Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
```

```
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }
```

Penjelasan:

Fungsi `deleteAll` akan menghapus seluruh elemen yang ada dalam Double Linked List.

- Mulai dari node pertama (`head`), iterasi melalui seluruh list menggunakan pointer `current`.
- Pada setiap node, simpan pointer ke node tersebut pada variabel sementara `temp`.
- Pindahkan `current` ke node selanjutnya.
- Hapus node yang disimpan pada `temp`.
- Setelah semua node terhapus, set `head` dan `tail` menjadi `nullptr` untuk menandakan list kosong.

7) Operasi Penampilan Elemen

```
// Fungsi untuk menampilkan semua elemen di list
    void display() {
        Node* current = head;
        cout << "List Tersimpan: ";
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};
```

Penjelasan:

Fungsi `display` akan mencetak semua nilai data yang tersimpan dalam Double Linked List.

- Mulai dari node pertama `(head)`, iterasi melalui seluruh list menggunakan pointer `current`.
- Pada setiap node, cetak nilai data pada node tersebut.
- Pindahkan `current`t ke node selanjutnya.
- Setelah semua node tercetak, tambahkan baris baru

8) Main

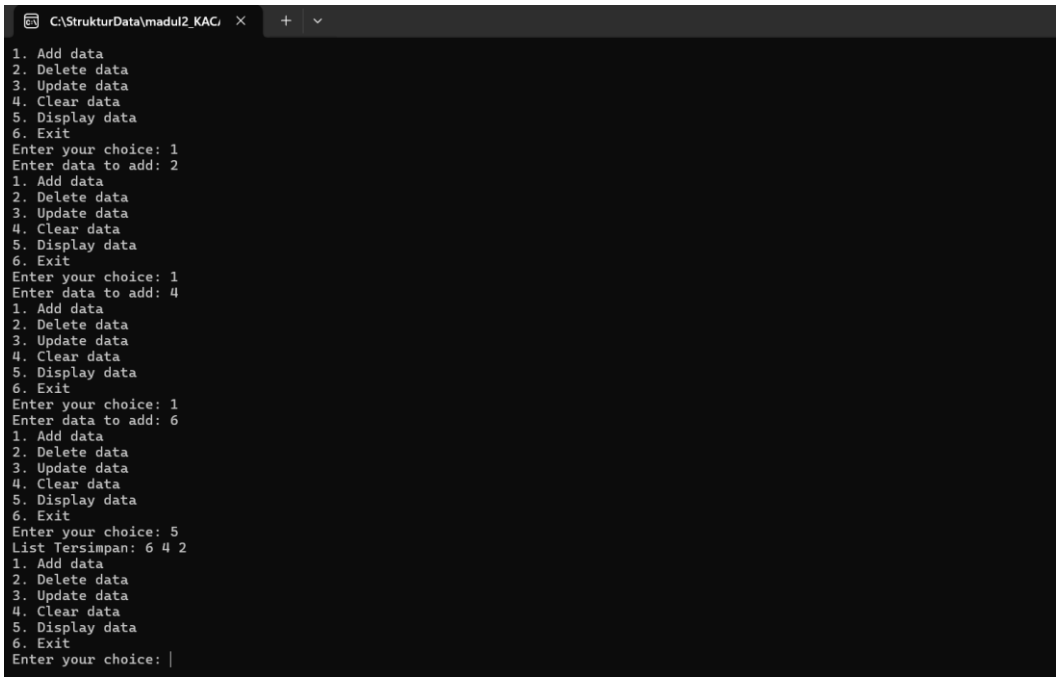
```
int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;
        int choice;
        cout << "Enter your choice: ";
```

```
        cin >> choice;
        switch (choice) {
            case 1: {
    …
    }
    return 0;
}
```

Dalam case ini akan ditampilkan menu berupa list pilihan yang akan dipilih oleh usr dari menu 1-6, setiap menu yang dipilih akan menjalankan operasi tertentu yang telah didefinisikan seperti di atas.

Output Guided:

Insert data (insert first) : 2, 4, 6



Delete Data (menghapus data pertama):

Update data: 4 > 6

```
Enter your choice: 5
List Tersimpan: 4 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 4
Enter new data: 6
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
List Tersimpan: 6 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

Clear data:

```
Enter your choice: 5
List Tersimpan: 6 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
List Tersimpan:
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
```

Exit:

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6

Process returned 0 (0x0)   execution time : 159.530 s
Press any key to continue.
```

5. Unguided

1.

1. Buatlah ADT *Double Linked list* sebagai berikut di dalam *file* **"doublelist.h"**:

```
Type infotype : kendaraan <
    nopol : string
    warna : string
    thnBuat : integer
>
Type address : pointer to ElmList
Type ElmList <
    info : infotype
    next :address
    prev : address
>

Type List <
    First : address
    Last : address
>
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertLast( in/out L : List, in P : address )
```

Buatlah implementasi ADT *Double Linked list* pada *file* "doublelist.cpp" dan coba hasil implementasi ADT pada *file* "main.cpp".

doublelist.h:

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

struct infotype {
    string nopol;
    string warna;
    int thnBuat;
};

typedef struct ElmList *address;

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

struct List {
    address First;
    address Last;
};

// Prosedur dan fungsi dasar
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertLast(List &L, address P);

#endif
```

doublelist.cpp:

```
#include "doublelist.h"
#include <iostream>

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) {
    address P = L.Last;  // Start from the last element
    cout << "DATA LIST (LIFO)" << endl;
    while (P != NULL) {
        cout << "No polisi : " << P->info.nopol << endl;
        cout << "Warna     : " << P->info.warna << endl;
        cout << "Tahun     : " << P->info.thnBuat << endl;
        P = P->prev;  // Move to the previous element
    }
}

void insertLast(List &L, address P) {
    if (L.First == NULL) {
        L.First = P;
        L.Last = P;
    } else {
        P->prev = L.Last;
        L.Last->next = P;
        L.Last = P;
    }
}
```

main.cpp

```
//#include "doublelist.h"
#include <iostream>

using namespace std;

int main() {
    List L;
    CreateList(L);
    infotype kendaraan;
    char again;

    // Input data
    do {
        cout << "Masukkan nomor polisi: ";
        cin >> kendaraan.nopol;
        cout << "Masukkan warna kendaraan: ";
        cin >> kendaraan.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> kendaraan.thnBuat;

        // Cek duplikasi nomor polisi
```

```
        if (findElm(L, kendaraan) != NULL) {
            cout << "Nomor polisi sudah terdaftar" << endl << endl;
            continue;
        }

        address P = alokasi(kendaraan);
        insertLast(L, P);

        cout << "\nTambah data lagi? (y/n): ";
        cin >> again;
    } while (again == 'y' || again == 'Y');

    cout << "\n";
    printInfo(L);

    return 0;
}
```

Output:

```
Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90

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

Tambah data lagi? (y/n): y
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

Tambah data lagi? (y/n): n

DATA LIST (LIFO)
No polisi : D004
Warna     : kuning
Tahun     : 90
No polisi : D003
Warna     : putih
Tahun     : 70
No polisi : D001
Warna     : hitam
Tahun     : 90

Process returned 0 (0x0)   execution time : 52.484 s
```

Deskripsi Program:

Fungsi utama dalam program ini meliputi:

- `CreateList`: Menginisialisasi daftar kosong.
- `alokasi` dan `dealokasi`: Mengalokasikan dan menghapus node untuk menyimpan data kendaraan.
- `insertLast`: Menambahkan node di akhir daftar.
- `printInfo`: Menampilkan data kendaraan dalam urutan LIFO (Last-In-First-Out), dimulai dari node terakhir hingga node pertama.

Pengguna dapat menambahkan data kendaraan melalui masukan konsol, dan program menampilkan seluruh data kendaraan dalam urutan LIFO setelah data dimasukkan.

Output: Program menampilkan data kendaraan dari node terakhir hingga node pertama, sesuai urutan LIFO.

2.

Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru.
fungsi findElm(L : *List*, x : infotype) : *address*

```
Masukkan Nomor Polisi yang dicari   : D001

Nomor Polisi   : D001
Warna          : hitam
Tahun          : 90
```

Gambar 6-24 *Output* mencari nomor polisi

doublelist.h:

```
 // Fungsi tambahan find
address findElm(List L, infotype x);
```

doublelist.cpp:

```
 // Fungsi tambahan
 address findElm(List L, infotype x) {
     address P = L.First;
     while (P != NULL) {
         if (P->info.nopol == x.nopol) {
             return P;
         }
         P = P->next;
     }
     return NULL;
}
```

main.cpp:

```
 // Fitur pencarian
     cout << "\nMasukkan Nomor Polisi yang dicari: ";
     cin >> kendaraan.nopol;

     address found = findElm(L, kendaraan);
     if (found != NULL) {
         cout << "Nomor Polisi : " << found->info.nopol << endl;
         cout << "Warna        : " << found->info.warna << endl;
         cout << "Tahun        : " << found->info.thnBuat << endl;
     } else {
         cout << "Data tidak ditemukan!" << endl;
    }
```

Output:

```
DATA LIST (LIFO)
No polisi : D004
Warna     : kuning
Tahun     : 90
No polisi : D003
Warna     : putih
Tahun     : 70
No polisi : D001
Warna     : hitam
Tahun     : 90

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

Deskripsi program:
Dalam program ini, fungsi tambahan `findElm` ditambahkan untuk pencarian data kendaraan berdasarkan nomor polisi.

Detail Fungsi:

- `findElm`: Mengembalikan alamat node yang memiliki nomor polisi yang cocok dengan input, atau NULL jika tidak ditemukan.
- Programdapat mendeteksi jika ada duplikasi nomor polisi sebelum menambahkan data kendaraan baru, sehingga nomor polisi unik terjamin dalam daftar.
- Fungsi ini akan mengembalikan alamat dari node yang sesuai dengan nomor polisi yang dicari, atau NULL jika data tidak ditemukan.

Output: Program akan menampilkan pesan peringatan jika nomor polisi yang dimasukkan sudah ada dalam daftar.

3.

Hapus elemen dengan nomor polisi D003 dengan prosedur *delete*.
- prosedur deleteFirst(in/out L : *List*, in/out P : *address*)
- prosedur deleteLast(in/out L : *List*, in/out P : *address*)
- prosedur deleteAfter(in Prec : *address*, in/out: P : *address*)

```
Masukkan Nomor Polisi yang akan dihapus   : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

Nomor Polisi   : D004
Warna          : kuning
Tahun          : 90
Nomor Polisi   : D001
Warna          : hitam
Tahun          : 90
```

Gambar 6-25 *Output* menghapus data nomor polisi

doublelist.h:

```
// Fungsi tambahan delete
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
 void deleteAfter(address Prec, address &P);
```

doublelist.cpp:

```
void deleteFirst(List &L, address &P) {
    if (L.First != NULL) {
        P = L.First;
        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 = 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(address Prec, address &P) {
    if (Prec != NULL) {
        P = Prec->next;
        if (P != NULL) {
            Prec->next = P->next;
            if (P->next != NULL) {
                P->next->prev = Prec;
            }
            P->next = NULL;
            P->prev = NULL;
        }
    }
}

void deleteData(List &L, string nopol) {
    infotype searchData;
    searchData.nopol = nopol;

    address P = findElm(L, searchData);
    if (P != NULL) {
        if (P == L.First) {
            deleteFirst(L, P);
        } else if (P == L.Last) {
            deleteLast(L, P);
        } else {
            deleteAfter(P->prev, P);
        }
        cout << "Data dengan nomor polisi " << nopol << " berhasil
   dihapus." << endl;
        dealokasi(P);
    } else {
        cout << "Data tidak ditemukan!" << endl;
    }
}
```

main.cpp:

```
// Fitur penghapusan
    cout << "\nMasukkan Nomor Polisi yang akan dihapus: ";
    string nopolHapus;
    cin >> nopolHapus;

    deleteData(L, nopolHapus);

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

Output:

```
Masukkan Nomor Polisi yang akan dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1
DATA LIST (LIFO)
No polisi : D004
Warna     : kuning
Tahun     : 90
No polisi : D001
Warna     : hitam
Tahun     : 90
```

Deskripsi program:

Program ini menyediakan fungsi `deleteData` untuk menghapus data kendaraan dari Doubly Linked List berdasarkan nomor polisi yang diinput pengguna. Fungsi ini menggunakan `findElm` untuk mencari data yang akan dihapus dan memanfaatkan prosedur `deleteFirst, deleteLast,` atau `deleteAfter` sesuai posisi data dalam daftar.

Rincian Proses:

- Input Nomor Polisi: Program meminta pengguna memasukkan nomor polisi kendaraan yang ingin dihapus.
- Pencarian Data: Fungsi `findElm` mencari data berdasarkan nomor polisi. Jika data ditemukan, posisi data dalam daftar akan ditentukan.
- Penghapusan Berdasarkan Posisi:
- `deleteFirst:` Jika data berada di posisi pertama.
- `deleteLast:` Jika data berada di posisi terakhir.
- `deleteAfter:` Jika data berada di tengah daftar.

Pesan Konfirmasi: Program menampilkan pesan bahwa data berhasil dihapus atau "Data tidak ditemukan!" jika nomor polisi tidak ada dalam daftar.

Output: Setelah penghapusan, program menampilkan daftar kendaraan yang tersisa dan pesan keberhasilan penghapusan atau pesan kesalahan jika data tidak ditemukan.

6. Kesimpulan

Melalui praktikum ini, mahasiswa memahami implementasi struktur data Doubly Linked List untuk pengelolaan data secara dinamis, di mana setiap node memiliki koneksi dua arah. Dengan menerapkan operasi dasar seperti penambahan, pencarian, dan penghapusan data, mahasiswa belajar bagaimana mengatur dan memanipulasi data dalam daftar berantai dengan lebih efisien.

Mahasiswa juga memahami pentingnya metode pencarian (findElm) untuk mendeteksi data yang duplikat atau yang akan dihapus, serta menampilkan data. Praktikum ini mengasah kemampuan dalam merancang dan mengimplementasikan fungsi sesuai kebutuhan, serta meningkatkan pemahaman tentang pengelolaan

memori melalui proses alokasi dan dealokasi.

Secara keseluruhan, praktikum ini memberikan dasar yang kuat dalam pemahaman Doubly Linked List, yang sangat berguna dalam berbagai aplikasi pemrograman yang memerlukan struktur data dinamis dan efisien.