Telkom University Purwokerto

Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

LAPORAN PRAKTIKUM
MODUL 6
DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :

Zaenarif Putra ‘Ainurdin – 2311104049

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

PROGRAM STUDI SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

I. TUJUAN

1. Memahai konsep modul linked list.
2. Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C

II. LANDASAN TEORI

1. Double Linked List

   Double Linked list adalah linked list yang masing – masing elemen nya memiliki 2 successor, yaitu successor yang menunjuk pada elemen sebelumnya (prev) dan successor yang menunjuk pada elemen sesudahnya (next).

2. Insert

   Insert First yaitu menyisipkan sebuah elemen baru di awal list. Elemen baru ini akan menjadi elemen pertama (first), dan elemen sebelumnya yang menjadi first akan digeser ke posisi berikutnya. Langkahnya melibatkan pengaturan pointer next pada elemen baru agar menunjuk ke elemen first yang lama dan memperbarui pointer prev dari elemen first yang lama untuk menunjuk ke elemen baru.

   Insert Last yaitu menyisipkan elemen baru di akhir list. Elemen baru ini akan menjadi elemen terakhir (last), dan elemen sebelumnya yang menjadi last akan menunjuk ke elemen baru. Langkah-langkahnya adalah mengatur pointer prev pada elemen baru untuk menunjuk ke elemen last yang lama, dan memperbarui next dari elemen last lama ke elemen baru.

   Insert After yaitu menyisipkan elemen baru setelah elemen tertentu dalam list. Elemen baru ditempatkan setelah elemen yang sudah ada, dengan mengatur pointer next dan prev sehingga elemen baru berada di antara dua elemen yang sudah ada. Metode ini memerlukan pencarian elemen tertentu terlebih dahulu sebelum penyisipan dilakukan.

   Insert Before yaitu menyisipkan elemen baru sebelum elemen tertentu dalam list. Prosesnya adalah kebalikan dari Insert After, dengan elemen baru ditempatkan sebelum elemen yang sudah ada. Ini melibatkan pencarian elemen tertentu terlebih dahulu, kemudian mengatur pointer next dan prev untuk elemen yang relevan.

3. Delete

   Delete First menghapus elemen pertama (first) dari list. Elemen first yang baru akan menjadi elemen setelah first yang lama. Pointer prev dari elemen first baru diatur menjadi NULL, dan elemen first lama dihapus atau dialokasikan ulang (dealokasi)

Delete Last menghapus elemen terakhir (last) dari list. Elemen last yang baru adalah elemen sebelum last yang lama. Pointer next dari elemen last yang baru diatur menjadi NULL, dan elemen last lama dihapus atau dialokasikan ulang.

Delete After menghapus elemen setelah elemen tertentu. Pertama, elemen yang ingin dihapus dicari berdasarkan posisi setelah elemen tertentu. Setelah ditemukan, pointer next dari elemen sebelum elemen yang dihapus dan pointer prev dari elemen setelahnya diatur untuk saling menunjuk, menghilangkan elemen yang akan dihapus dari daftar.

Delete Before menghapus elemen sebelum elemen tertentu. Metode ini adalah kebalikan dari Delete After, di mana elemen yang akan dihapus adalah elemen sebelum elemen yang ditunjuk. Setelah elemen ditemukan, pointer next dan prev diatur untuk saling menunjuk, menghapus elemen dari list.

Dalam Double Linked List, Update, View, dan Searching dapat dilakukan secara terpadu. Proses ini dimulai dengan pencarian elemen tertentu berdasarkan kriteria tertentu, yang bisa dilakukan dari first atau last untuk fleksibilitas. Setelah elemen ditemukan, data pada elemen tersebut bisa langsung ditampilkan untuk verifikasi (View) atau diperbarui jika ada informasi yang perlu diubah (Update). Dengan demikian, ketiga operasi ini dapat berjalan secara berkesinambungan, memudahkan manipulasi dan pengelolaan data dalam list tanpa perlu melakukan pencarian ulang.

III. GUIDE

1. Guide6M

   a. Syntax

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

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

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

    DoublylinkedList()  {
        head = nullptr;
        tail = nullptr;
    }

    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;
        }
        head = newNode;
    }

    void deleteNode() {
        if(head == nullptr) {
            return;
        }
        Node* temp = head;
        head = head->next;
        if(head != nullptr) {
            head->prev = nullptr;
        }
        else {
            tail = nullptr;
        }
        delete temp;
    }

```

```
    bool update(int oldData, int newData) {
        Node* current = head;
        while(current != nullptr) {
            if(current->data == oldData) {
                current->data = newData;
                return true;
            }
            current = current->next;
        }
        return false;
    }

    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

```

```
int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl; //Menampilkan Penambahan Data
        cout << "2. Delete data" << endl; //Menampilkan Penghapusan Data
        cout << "3. Update data" << endl; //Menampilkan Pembaruan Data
        cout << "4. Clear data" << endl; //Menampilkan Pembersihan Data
        cout << "5. Display data" << endl; //Menampilkan Data Yang Sudah Di Masukkan
        cout << "6. Exit" << endl; //Menampilkan Keluar Dari Program

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
                list.insert(data);
                break;
            }
            case 2: {
                list.deleteNode();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                bool updated = list.update(oldData, newData);
                if (!updated) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
    return 0;
}
```

b. Penjelasan syntax

- Class Node :

    - Yang dimana merupakan struktur dasar dari setiap node dalam Double Linked List, dengan atribut *data* (untuk menyimpan nilai), *prev* (pointer menuju node sebelumnya), dan *next* (pointer menuju node berikutnya).

- Class Double Linked List :

    - Terdapat *constructor* yang digunakan untuk menginisialisasi *head* dan *tail* dengan *nullptr* untuk memulai list yang kosong.
    - Insert(int data) yaitu membuat node baru yang dibuat dengan data yang dimasukkan oleh user. Jika list masih kosong, node baru akan menjadi *head* dan *tail*. Jika tidak kosong, *prev* dari *head* yang lama akan diatur ke node yang baru.
    - deleteNode() digunakan sebagai penghapus elemen pertama dari list. Jika list kosong, fungsi keluar tanpa melakukan apapun. Jika tidak, *head* diatur ke elemen berikutnya, dan *prev* dari *head* baru diatur ke *nullptr*.
    - update(int oldData, int newData) Digunakan untuk mencari data tertentu (*oldData*) di list. Jika ditemukan, nilai data diubah menjadi newData. Fungsi ini mengembalikan true jika pembaruan berhasil, dan *false* jika data tidak ditemukan.
    - deleteAll() Digunakan untuk menghapus seluruh elemen dari list dengan mengiterasi setiap node, menghapusnya satu per satu, dan akhirnya mengatur *head* dan *tail* ke *nullptr*
    - display() Digunakan untuk menampilkan semua data dalam list dengan mengiterasi dari *head* hingga *tail*.

- main() Function :

    - Menyediakan menu pilihan interaktif untuk user dengan isi sebagai berikut :
        - Menambah data (menyisipkan di awal list).
        - Menghapus data (elemen pertama dihapus).
        - Memperbarui data (mengubah data lama menjadi data baru jika ditemukan).
        - Menghapus seluruh data di list.
        - Menampilkan data dalam list.
        - Keluar dari program.
    - switch digunakan untuk menangani pilihan user, untuk menjalankan metode yang sesuai berdasarkan input pengguna.

c. Output

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
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: 2
Enter new data: 10
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
```

```
Enter your choice: 5
10
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 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

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan6\guide\output>
```

IV. UNGUIDED

1. Task1

   a. Syntax “doublelist.h”

```h
#ifndef DOUBLIST_H
#define DOUBLIST_H
#include <string>
using namespace std;

class Vehicle {
public:
    string nopol;
    string warna;
    int tahun;
    Vehicle* next;
    Vehicle* prev;

    Vehicle(string nopol, string warna, int tahun);
};

class VehicleList {
private:
    Vehicle* first;
    Vehicle* last;

public:
    VehicleList();
    ~VehicleList();

    void insertLast(Vehicle* P);
    void insertAtPosition(int position, Vehicle* P);
    Vehicle* searchVehicle(string nopol);
    void deleteVehicle(string nopol);
    void printList() const;
    bool isRegistered(string nopol) const;
};

#endif

```

Syntax “doublelist.cpp”

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

Vehicle::Vehicle(string nopol, string warna, int tahun) {
    this->nopol = nopol;
    this->warna = warna;
    this->tahun = tahun;
    this->next = nullptr;
    this->prev = nullptr;
}

VehicleList::VehicleList() {
    first = nullptr;
    last = nullptr;
}

VehicleList::~VehicleList() {

}

void VehicleList::insertLast(Vehicle* P) {
    if (first == nullptr) {
        first = P;
        last = P;
    } else {
        last->next = P;
        P->prev = last;
        last = P;
    }
}

```

Telkom University Purwokerto

```cpp
void VehicleList::insertAtPosition(int position, Vehicle* P) {
    if (position == 1) {
        if (first == nullptr) {
            first = P;
            last = P;
        } else {
            P->next = first;
            first->prev = P;
            first = P;
        }
    } else {
        Vehicle* current = first;
        int index = 1;
        while (current != nullptr && index < position - 1) {
            current = current->next;
            index++;
        }
        if (current == nullptr || current->next == nullptr) {
            insertLast(P);
        } else {
            P->next = current->next;
            P->prev = current;
            current->next->prev = P;
            current->next = P;
        }
    }
}

Vehicle* VehicleList::searchVehicle(string nopol) {
    Vehicle* P = first;
    while (P != nullptr) {
        if (P->nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}
```

```cpp
void VehicleList::deleteVehicle(string nopol) {
    Vehicle* current = first;
    while (current != nullptr) {
        if (current->nopol == nopol) {
            if (current == first) {
                first = current->next;
                if (first != nullptr) first->prev = nullptr;
            } else if (current == last) {
                last = current->prev;
                if (last != nullptr) last->next = nullptr;
            } else {
                current->prev->next = current->next;
                current->next->prev = current->prev;
            }
            delete current;
            return;
        }
        current = current->next;
    }
}

void VehicleList::printList() const {
    Vehicle* P = first;
    int i = 1;
    while (P != nullptr) {
        cout << "DATA LIST " << i << endl;
        cout << "Nomor polisi : " << P->nopol << endl;
        cout << "Warna      : " << P->warna << endl;
        cout << "Tahun      : " << P->tahun << endl;
        P = P->next;
        i++;
    }
}

bool VehicleList::isRegistered(string nopol) const {
    Vehicle* P = first;
    while (P != nullptr) {
        if (P->nopol == nopol) {
            return true;
        }
        P = P->next;
    }
    return false;
}
```

Syntax “main.cpp”

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

using namespace std;

int main() {
    VehicleList L;
    string nopol, warna;
    int tahun;
    int numVehicles;

    cout << "Berapa kendaraan yang ingin Anda masukkan? ";
    cin >> numVehicles;

    for (int i = 1; i <= numVehicles; ++i) {
        bool validNopol = false;

        while (!validNopol) {
            cout << "masukkan nomor polisi (kendaraan " << i << "): ";
            cin >> nopol;

            if (L.isRegistered(nopol)) {
                Vehicle* registeredVehicle = L.searchVehicle(nopol);
                cout << "Nomor polisi sudah terdaftar dengan detail berikut:" << endl;
                cout << "Nomor polisi : " << registeredVehicle->nopol << endl;
                cout << "Warna        : " << registeredVehicle->warna << endl;
                cout << "Tahun        : " << registeredVehicle->tahun << endl;
                cout << "Silakan masukkan nomor polisi yang berbeda." << endl;
            } else {
                validNopol = true;
            }
        }

        cout << "masukkan warna kendaraan: ";
        cin >> warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> tahun;
        cout << "" << endl;

        Vehicle* P = new Vehicle(nopol, warna, tahun);
        L.insertLast(P);
    }

    cout << "\nDaftar kendaraan yang berhasil dimasukkan:\n";
    L.printList();

    return 0;
}
```

b. Penjelasan Syntax

* Syntax doubelist.h

Syntax berikut mendefinisikan dua kelas utama, *Vehicle* dan *VehicleList*, untuk mengelola data kendaraan dalam sebuah *Double Linked List*.

Class *vehicle* digunakan untuk menyimpan data mengenai kendaraan, yang dimana terdapat beberapa atribut antara lain :

- nopol: Nomor polisi kendaraan.
- warna: Warna kendaraan.
- tahun: Tahun pembuatan kendaraan.
- next: Pointer ke node berikutnya (untuk list berantai ganda).
- prev: Pointer ke node sebelumnya (untuk list berantai ganda).

Kemudian untuk constructor Vehicle(string nopol, string warna, int tahun) : digunakan untuk menginisialisasi atribut nopol, warna, dan tahun untuk objek Vehicle.

Class vehiclelist digunakan untuk mengelola objek *Vehicle* yang mempresentasikan daftar kendaraan. Kemudian untuk atributnya sebagai berikut :

- first: Pointer ke node pertama dalam list.
- last: Pointer ke node terakhir dalam list.

Untuk Constructor *VehicleList()* : digunakan untuk menginisialisasikan *fisrt* dan *last* dengan *nullptr* untuk memulai list kosong.

Untuk Destructor *~VehicleList() :* Menghapus list saat objek VehicleList dihapus dari memori.

Method insertLast(Vehicle* P): Menambahkan objek Vehicle baru (P) di akhir list.

Method insertAtPosition(int position, Vehicle* P): Menyisipkan objek Vehicle baru (P) pada posisi tertentu dalam list berdasarkan indeks posisi.

Method searchVehicle(string nopol): Mencari kendaraan berdasarkan nomor polisi (nopol) dan mengembalikan pointer ke Vehicle jika ditemukan; mengembalikan nullptr jika tidak ditemukan.

Method deleteVehicle(string nopol): Menghapus kendaraan dari list berdasarkan nomor polisi (nopol).

Method printList() const: Menampilkan seluruh data kendaraan dalam list ke layar.

Method isRegistered(string nopol) const: Memeriksa apakah kendaraan dengan nomor polisi tertentu sudah terdaftar di list, mengembalikan true jika ditemukan dan false jika tidak.

* Syntax doublelist.cpp

Constructor Vehicle::Vehicle: Menginisialisasi objek Vehicle dengan nomor polisi (nopol), warna (warna), dan tahun pembuatan (tahun) yang diberikan. Pointer next dan prev diatur ke nullptr untuk mengindikasikan bahwa node ini tidak memiliki tetangga pada awalnya.

Constructor VehicleList::VehicleList: Menginisialisasi objek VehicleList dengan first dan last diset ke nullptr, menandakan list kosong.

Destructor VehicleList::~VehicleList: Menghapus list ketika objek VehicleList keluar dari cakupan, tetapi implementasinya kosong di sini, sehingga tidak ada tindakan penghapusan otomatis untuk setiap node dalam list.

Method insertLast: Menambahkan node P ke akhir list. Jika list kosong (first kosong), P akan menjadi elemen pertama dan terakhir. Jika tidak, P ditambahkan setelah elemen terakhir (last), dan pointer last diperbarui ke P.

Method insertAtPosition: Menyisipkan node P pada posisi tertentu dalam list. Jika position adalah 1, P ditambahkan di awal list. Jika posisi lain, loop

digunakan untuk menemukan posisi yang diinginkan, kemudian P disisipkan di sana. Jika posisi lebih besar dari panjang list, P ditempatkan di akhir dengan memanggil insertLast.

Method searchVehicle: Mencari kendaraan dalam list berdasarkan nomor polisi (nopol). Jika ditemukan, mengembalikan pointer ke node tersebut. Jika tidak ditemukan, mengembalikan nullptr.

Method deleteVehicle: Menghapus node dari list berdasarkan nomor polisi (nopol). Jika node yang ditemukan adalah first, maka first diperbarui ke node berikutnya. Jika node adalah last, last diperbarui ke node sebelumnya. Jika berada di tengah, pointer next dan prev dari tetangga diatur untuk saling menunjuk, menghapus node tersebut dari list.

Method printList: Menampilkan seluruh data kendaraan dalam list dengan loop dari first hingga last, mencetak nomor polisi, warna, dan tahun kendaraan.

Method isRegistered: Mengecek apakah ada kendaraan dengan nomor polisi tertentu dalam list. Mengembalikan true jika ditemukan, dan false jika tidak.

* Syntax main.cpp

Deklarasi dan Inisialisasi: Program dimulai dengan mendeklarasikan objek VehicleList L untuk menyimpan daftar kendaraan, variabel untuk menyimpan input pengguna (nopol, warna, tahun), dan numVehicles sebagai jumlah kendaraan yang ingin dimasukkan oleh pengguna.

Meminta Jumlah Kendaraan: Program meminta input pengguna untuk menentukan jumlah kendaraan (numVehicles) yang akan dimasukkan ke dalam daftar.

Loop untuk Memasukkan Kendaraan: Loop for digunakan untuk mengulangi proses memasukkan data kendaraan sebanyak numVehicles kali.

Validasi Nomor Polisi: Dalam setiap iterasi, program melakukan validasi apakah nomor polisi (nopol) yang dimasukkan pengguna sudah ada dalam daftar kendaraan (VehicleList).

isRegistered: Memanggil isRegistered(nopol) untuk memeriksa keberadaan nomor polisi dalam list.
Jika nomor polisi sudah ada, program mengambil data kendaraan tersebut menggunakan searchVehicle(nopol) dan menampilkan detail kendaraan yang sudah terdaftar, kemudian meminta pengguna untuk memasukkan nomor polisi yang berbeda.

Jika nomor polisi tidak ada dalam list, validNopol diatur ke true untuk melanjutkan ke langkah berikutnya.
Input Detail Kendaraan: Setelah nomor polisi terverifikasi, program

meminta pengguna memasukkan warna dan tahun kendaraan.

Membuat dan Menyisipkan Node Kendaraan:

Membuat objek Vehicle baru (P) dengan constructor Vehicle(nopol, warna, tahun) menggunakan data yang dimasukkan.
Menyisipkan kendaraan baru ke akhir daftar menggunakan L.insertLast(P).

Menampilkan Daftar Kendaraan: Setelah semua data kendaraan berhasil dimasukkan, program menampilkan daftar kendaraan dengan memanggil printList(), yang menampilkan nomor polisi, warna, dan tahun dari setiap kendaraan dalam list.

c. Output

```
Berapa kendaraan yang ingin Anda masukkan? 3
masukkan nomor polisi (kendaraan 1): D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi (kendaraan 2): D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi (kendaraan 3): D001
Nomor polisi sudah terdaftar dengan detail berikut:
Nomor polisi : D001
Warna        : hitam
Tahun        : 90
Silakan masukkan nomor polisi yang berbeda.
masukkan nomor polisi (kendaraan 3): D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90


Daftar kendaraan yang berhasil dimasukkan:
DATA LIST 1
Nomor polisi : D001
Warna     : hitam
Tahun     : 90
DATA LIST 2
Nomor polisi : D003
Warna     : putih
Tahun     : 70
DATA LIST 3
Nomor polisi : D004
Warna     : kuning
Tahun     : 90
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan6\unguided\output>
```

2. Task2 elemen dengan nomor polisi D001 dengan membuat fungsi baru.

a. Syntax

```
case 2: {
    cout << "Masukkan nomor polisi kendaraan yang ingin dicari: ";
    cin >> nopol;
    Vehicle* foundVehicle = L.searchVehicle(nopol);
    if (foundVehicle) {
        cout << "Detail kendaraan ditemukan:" << endl;
        cout << "Nomor polisi : " << foundVehicle->nopol << endl;
        cout << "Warna        : " << foundVehicle->warna << endl;
        cout << "Tahun        : " << foundVehicle->tahun << endl;
    } else {
        cout << "Kendaraan dengan nomor polisi tersebut tidak ditemukan." << endl;
    }
    break;
}
```

b. Penjelasan Syntax

cout << "Masukkan nomor polisi kendaraan yang ingin dicari: "; Menampilkan pesan ke layar yang meminta pengguna untuk memasukkan nomor polisi kendaraan yang ingin dicari.

cin >> nopol; Menerima input dari pengguna dan menyimpannya dalam variabel nopol, yang merupakan nomor polisi kendaraan yang akan dicari.

Vehicle* foundVehicle = L.searchVehicle(nopol); Fungsi searchVehicle dari objek L (yang merupakan list kendaraan) dipanggil untuk mencari kendaraan dengan nomor polisi yang dimasukkan (nopol). Jika kendaraan ditemukan, fungsi ini akan mengembalikan pointer ke kendaraan yang ditemukan, dan jika tidak, akan mengembalikan nullptr.

if (foundVehicle) Memeriksa apakah kendaraan ditemukan. Jika pointer foundVehicle tidak nullptr, berarti kendaraan ditemukan.

cout << "Detail kendaraan ditemukan:" << endl; Menampilkan pesan bahwa detail kendaraan ditemukan.

cout << "Nomor polisi : " << foundVehicle->nopol << endl; Menampilkan nomor polisi kendaraan yang ditemukan.

cout << "Warna : " << foundVehicle->warna << endl; Menampilkan warna kendaraan yang ditemukan.

cout << "Tahun : " << foundVehicle->tahun << endl; Menampilkan tahun kendaraan yang ditemukan.

else { cout << "Kendaraan dengan nomor polisi tersebut tidak ditemukan." << endl; } Jika kendaraan tidak ditemukan (yaitu foundVehicle adalah nullptr), maka menampilkan pesan bahwa kendaraan tersebut tidak ada dalam daftar.

break; Menghentikan eksekusi case ini dan kembali ke menu utama setelah pencarian selesai.

c. Output

```
=== Menu List Kendaraan ===
1. Tambah kendaraan
2. Cari kendaraan
3. Tampilkan semua kendaraan
4. Hapus kendaraan
5. Keluar
Pilih opsi: 2
Masukkan nomor polisi kendaraan yang ingin dicari: D001
Detail kendaraan ditemukan:
Nomor polisi : D001
Warna        : hitam
Tahun        : 90
```

3. Task3 jumlah total info seluruh elemen

   a. Syntax

```
case 4: {
                cout << "Masukkan nomor polisi kendaraan yang ingin dihapus: ";
                cin >> nopol;
                bool success = L.deleteVehicle(nopol);
                if (success) {
                    cout << "Kendaraan dengan nomor polisi " << nopol << " berhasil dihapus." << endl;
                } else {
                    cout << "Kendaraan dengan nomor polisi tersebut tidak ditemukan." << endl;
                }
                break;
```

   b. Penjelasan Syntax

   cout << "Masukkan nomor polisi kendaraan yang ingin dihapus: "; Menampilkan pesan ke layar yang meminta pengguna untuk memasukkan nomor polisi kendaraan yang ingin dihapus.

   cin >> nopol; Menerima input dari pengguna dan menyimpannya dalam variabel nopol, yang berisi nomor polisi kendaraan yang ingin dihapus.

   bool success = L.deleteVehicle(nopol); Memanggil fungsi deleteVehicle dari objek L (yang merupakan daftar kendaraan), untuk mencoba menghapus kendaraan yang memiliki nomor polisi yang dimasukkan (nopol). Fungsi deleteVehicle akan mengembalikan true jika kendaraan berhasil dihapus dan false jika kendaraan dengan nomor polisi tersebut tidak ditemukan.

   if (success) Memeriksa hasil dari fungsi deleteVehicle. Jika success bernilai true, berarti kendaraan berhasil dihapus.

   cout << "Kendaraan dengan nomor polisi " << nopol << " berhasil dihapus." << endl; Menampilkan pesan bahwa kendaraan dengan nomor polisi yang dimasukkan telah berhasil dihapus dari daftar.

   else { cout << "Kendaraan dengan nomor polisi tersebut tidak ditemukan." << endl; } Jika success bernilai false, berarti kendaraan dengan nomor polisi yang dimasukkan tidak ditemukan dalam daftar, dan akan ditampilkan pesan bahwa kendaraan tersebut tidak ditemukan.

   break; Menghentikan eksekusi dari case ini dan kembali ke menu utama setelah proses penghapusan selesai.

c. Output

```
=== Menu List Kendaraan ===
1. Tambah kendaraan
2. Cari kendaraan
3. Tampilkan semua kendaraan
4. Hapus kendaraan
5. Keluar
Pilih opsi: 4
Masukkan nomor polisi kendaraan yang ingin dihapus: D003
Kendaraan dengan nomor polisi D003 berhasil dihapus.

=== Menu List Kendaraan ===
1. Tambah kendaraan
2. Cari kendaraan
3. Tampilkan semua kendaraan
4. Hapus kendaraan
5. Keluar
Pilih opsi: 3

Daftar kendaraan yang berhasil dimasukkan:
DATA LIST 1
Nomor polisi : D001
Warna      : hitam
Tahun      : 90
DATA LIST 2
Nomor polisi : D004
Warna      : kuning
Tahun      : 90
```

V. KESIMPULAN

Praktikum Modul 6 membahas Double Linked List (DLL), yaitu struktur data yang memungkinkan setiap elemen memiliki dua pointer (next dan prev) untuk menunjuk ke elemen sebelum dan sesudahnya, sehingga memudahkan traversal data secara dua arah. Dengan komponen utama first (elemen pertama) dan last (elemen terakhir), DLL memungkinkan penambahan, penghapusan, dan pencarian elemen secara lebih efisien. Praktikum ini juga meliputi latihan implementasi operasi dasar seperti Insert dan Delete di berbagai posisi, serta pembuatan ADT untuk data kendaraan yang berisi atribut seperti nopol, warna, dan thnBuat.