LAPORAN PRAKTIKUM

MODUL 6

DOUBLE LINKED LIST (BAGIAN 2)

Nama :

Candra Dinata (2311104061)

Kelas :

S1SE 07 02

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Membantu mahasiswa memahami dasar-dasar pemrograman, seperti sintaks, tipe data, operator, dan struktur kontrol, serta menguasai penggunaan fungsi untuk modularitas program. Selain itu, mahasiswa juga diperkenalkan dengan konsep Object-Oriented Programming (OOP) seperti kelas, objek, pewarisan, dan enkapsulasi. Melalui latihan ini, mahasiswa diharapkan mampu menerapkan konsep pemrograman untuk menyelesaikan masalah dan memahami carakerja kompilator C++ dalam mengubah kode sumber menjadi program yang dapat dijalankan.

II. LANDASAN TEORI

C++ adalah bahasa pemrograman yang dikembangkan sebagai pengembangan dari bahasa C dengan menambahkan fitur-fitur pemrograman berorientasi objek (Object- Oriented Programming, OOP). Dalam C++, konsep OOP seperti enkapsulasi, pewarisan, dan polimorfis memenjadi dasar untuk membuat program yang lebih modular, terstruktur, dan dapat digunakan kembali. Bahasa ini mendukung penggunaan fungsi, manipulasi memori secara langsung melalui pointer, serta memiliki kemampuan untuk menangani berbagai tipe data dasar dan struktural. C++ juga mendukung konsep pemrograman prosedural dan modular, yang memudahkan pengembangan program yang kompleks. Kompilasi program C++ dilakukan melalui kompilator yang menerjemahkan kode sumber menjadi fileeksekusi yang bisa dijalankan oleh komputer.

III. GUIDED

1.

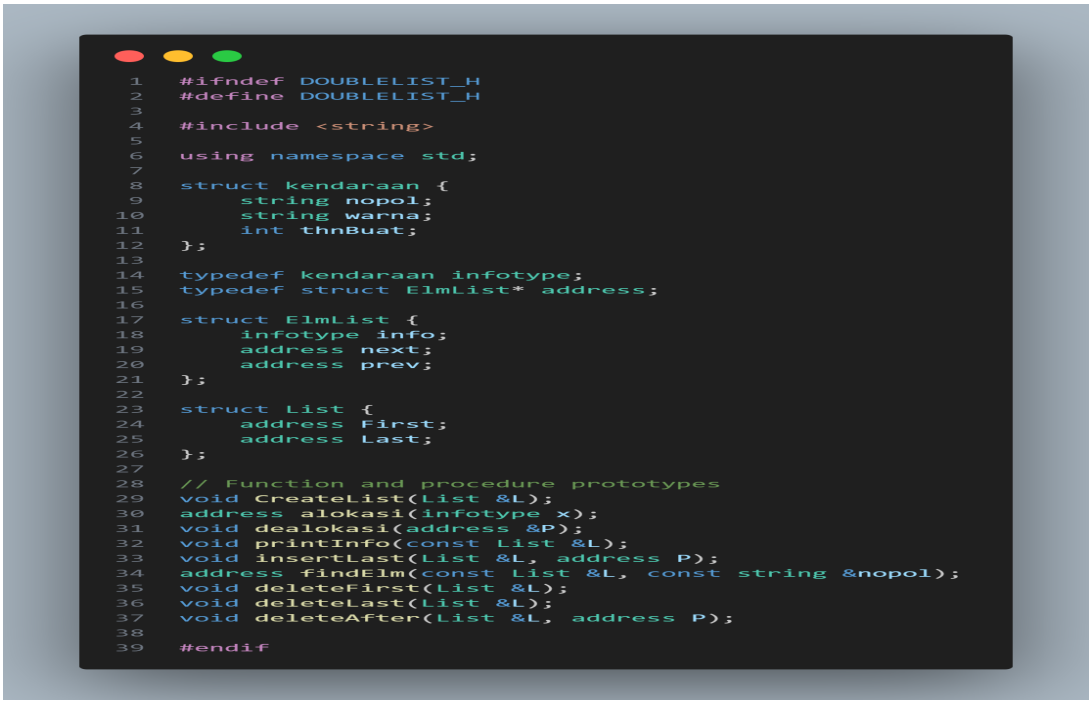
```cpp
#include <iostream>
using namespace std;

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

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

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

    // 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;
    }

    // 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
    }

    // 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
    }

    // 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;
    }

    // Fungsi untuk menampilkan semua elemen di list
    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;
        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: {
                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;
}
```

Kode ini mengimplementasikan struktur data Doubly Linked List (DLL) menggunakan kelas DoublyLinkedList dalam C++. Di dalam kelas ini, terdapat dua pointer, head dan tail, untuk menunjukkan awal dan akhir dari list. Kelas ini memiliki berbagai metode seperti insert, deleteNode, update, deleteAll, dan display untuk memanipulasi data di dalam list. Metode insert digunakan untuk menambahkan elemen baru di awal list. Jika list kosong, node baru tersebut akan menjadi head sekaligus tail. Metode deleteNode akan menghapus elemen di bagian depan list dan menyesuaikan pointer head dan tail jika list hanya memiliki satu elemen. Untuk mengupdate nilai dari suatu node, metode update mencari nilai lama (oldData) di dalam list dan menggantinya dengan nilai baru (newData). Jika nilai yang dicari tidak ditemukan, metode ini akan mengembalikan false.

Program ini memiliki antarmuka sederhana berbasis teks pada fungsi main(), yang menggunakan loop untuk menampilkan menu pilihan bagi pengguna. Setiap pilihan dalam menu ini memungkinkan pengguna untuk menambah, menghapus, mengupdate, membersihkan, atau menampilkan data dalam list. Pilihan menu ini diproses dengan menggunakan switch berdasarkan input dari pengguna. Jika pengguna memilih untuk menambahkan data, kode meminta pengguna memasukkan nilai baru dan menambahkannya di depan list. Pengguna juga dapat memilih untuk menghapus elemen terdepan, mengupdate elemen tertentu, menghapus semua elemen, atau menampilkan semua elemen yang ada di dalam list. Program akan berhenti ketika pengguna memilih opsi "Exit". Dengan adanya fungsi-fungsi tersebut, kode ini memberikan cara efektif untuk mengelola data secara dinamis dalam bentuk linked list yang memiliki akses ke elemen sebelumnya dan berikutnya melalui pointer prev dan next.

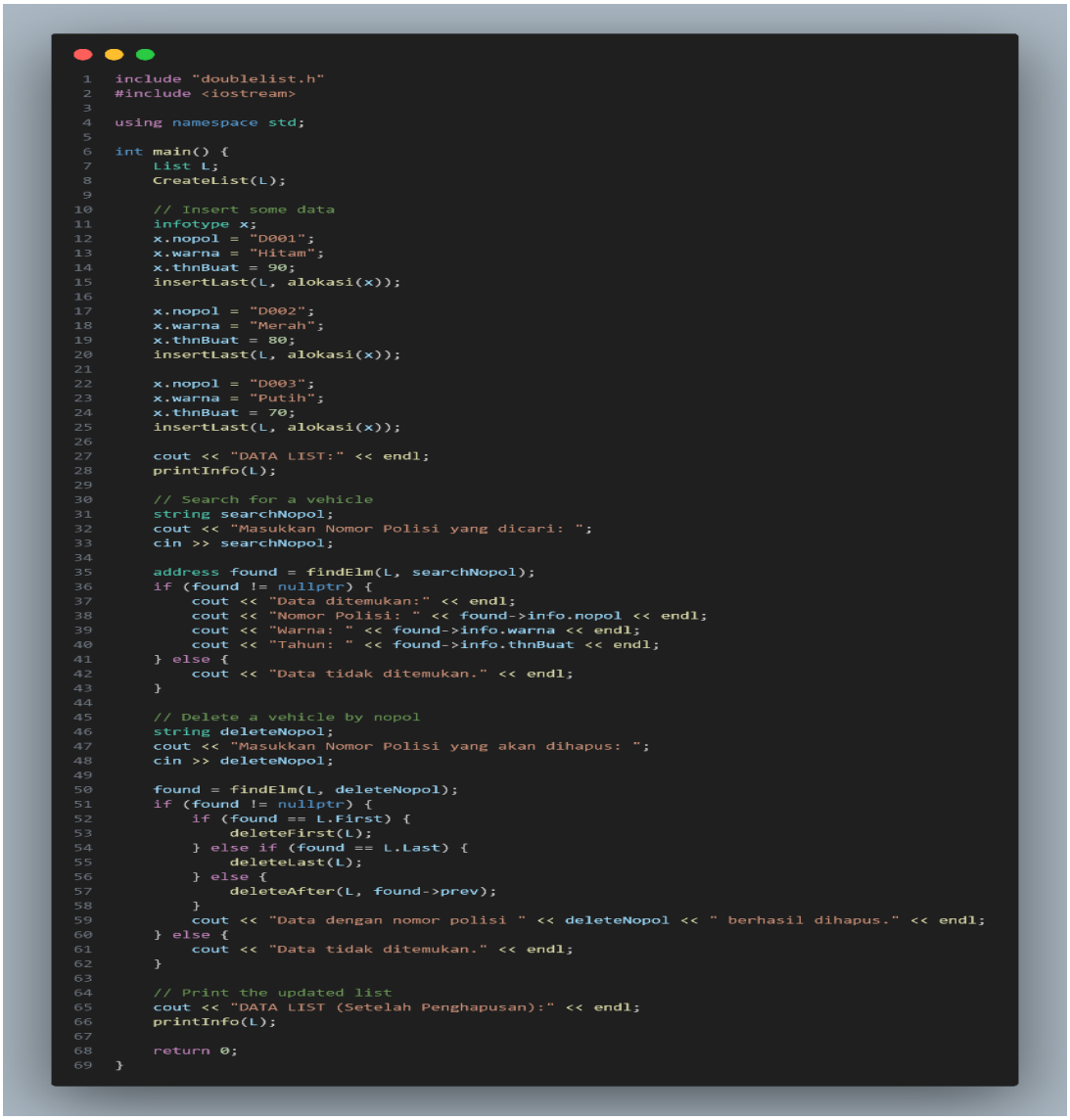
IV. UNGUIDED

1.



```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>

using namespace std;

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

typedef kendaraan infotype;
typedef struct ElmList* address;

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

struct List {
    address First;
    address Last;
};

// Function and procedure prototypes
void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertLast(List &L, address P);
address findElm(const List &L, const string &nopol);
void deleteFirst(List &L);
void deleteLast(List &L);
void deleteAfter(List &L, address P);

#endif
```

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

using namespace std;

// Initialize the list
void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

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

// Deallocate an element
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

// Insert an element at the end of the list
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;
    }
}

// Print all elements in the list
void printInfo(const List &L) {
    address P = L.First;
    while (P != nullptr) {
        cout << "Nomor Polisi: " << P->info.nopol << endl;
        cout << "Warna: " << P->info.warna << endl;
        cout << "Tahun: " << P->info.thnBuat << endl << endl;
        P = P->next;
    }
}

// Find an element by nopol
address findElm(const List &L, const string &nopol) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

// Delete the first element
void deleteFirst(List &L) {
    if (L.First != nullptr) {
        address P = L.First;
        if (L.First == L.Last) {
            L.First = nullptr;
            L.Last = nullptr;
        } else {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        dealokasi(P);
    }
}

// Delete the last element
void deleteLast(List &L) {
    if (L.Last != nullptr) {
        address P = L.Last;
        if (L.First == L.Last) {
            L.First = nullptr;
            L.Last = nullptr;
        } else {
            L.Last = L.Last->prev;
            L.Last->next = nullptr;
        }
        dealokasi(P);
    }
}

// Delete a specific element after a given element
void deleteAfter(List &L, address P) {
    if (P != nullptr && P->next != nullptr) {
        address Q = P->next;
        P->next = Q->next;
        if (Q->next != nullptr) {
            Q->next->prev = P;
        } else {
            L.Last = P;
        }
        dealokasi(Q);
    }
}
```



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

using namespace std;

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

    // Insert some data
    infotype x;
    x.nopol = "D001";
    x.warna = "Hitam";
    x.thnBuat = 90;
    insertLast(L, alokasi(x));

    x.nopol = "D002";
    x.warna = "Merah";
    x.thnBuat = 80;
    insertLast(L, alokasi(x));

    x.nopol = "D003";
    x.warna = "Putih";
    x.thnBuat = 70;
    insertLast(L, alokasi(x));

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

    // Search for a vehicle
    string searchNopol;
    cout << "Masukkan Nomor Polisi yang dicari: ";
    cin >> searchNopol;

    address found = findElm(L, searchNopol);
    if (found != nullptr) {
        cout << "Data ditemukan:" << endl;
        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;
    }

    // Delete a vehicle by nopol
    string deleteNopol;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> deleteNopol;

    found = findElm(L, deleteNopol);
    if (found != nullptr) {
        if (found == L.First) {
            deleteFirst(L);
        } else if (found == L.Last) {
            deleteLast(L);
        } else {
            deleteAfter(L, found->prev);
        }
        cout << "Data dengan nomor polisi " << deleteNopol << " berhasil dihapus." << endl;
    } else {
        cout << "Data tidak ditemukan." << endl;
    }

    // Print the updated list
    cout << "DATA LIST (Setelah Penghapusan):" << endl;
    printInfo(L);

    return 0;
}
```

Kode di atas adalah implementasi dari sebuah Doubly Linked List untuk menyimpan data kendaraan dengan informasi berupa nomor polisi, warna, dan tahun pembuatan. Kode ini dipecah ke dalam tiga file: doublelist.h, doublelist.cpp, dan main.cpp. File doublelist.h berfungsi sebagai header yang mendeklarasikan struktur data dan fungsi-fungsi utama yang digunakan. Di dalamnya, terdapat struktur kendaraan sebagai tipe data infotype yang menyimpan atribut kendaraan. Kemudian, ElmList adalah struktur yang mewakili elemen dalam list dengan pointer next dan prev untuk menunjukkan elemen sebelum dan sesudahnya. Struktur List mengandung dua pointer First dan Last yang menunjukkan elemen pertama dan terakhir dalam list.

File doublelist.cpp adalah implementasi dari fungsi-fungsi yang dideklarasikan di header. Fungsi CreateList menginisialisasi list dengan membuat First dan Last menjadi nullptr. Fungsi alokasi digunakan untuk mengalokasikan memori baru bagi elemen list, sedangkan dealokasi berfungsi untuk menghapus elemen dan membebaskan memori. Fungsi insertLast menambahkan elemen baru di akhir list. Jika list kosong, elemen baru menjadi First dan Last, namun jika sudah ada elemen, maka elemen baru akan ditambahkan setelah elemen terakhir. Fungsi printInfo menampilkan semua elemen dalam list, sementara findElm mencari elemen tertentu berdasarkan nomor polisi. Fungsi deleteFirst, deleteLast, dan

deleteAfter menghapus elemen dari posisi tertentu di list, yaitu dari awal, akhir, atau setelah elemen tertentu.

File main.cpp berisi implementasi main untuk menguji fungsionalitas Doubly Linked List ini. Pertama, list diinisialisasi menggunakan CreateList. Beberapa data kendaraan kemudian ditambahkan menggunakan fungsi insertLast untuk mengisi list dengan beberapa elemen. Data yang ada di list ditampilkan menggunakan printInfo, yang menunjukkan setiap elemen list dengan informasi nomor polisi, warna, dan tahun pembuatan. Program kemudian meminta pengguna untuk mencari kendaraan berdasarkan nomor polisi menggunakan findElm. Jika kendaraan ditemukan, program menampilkan datanya; jika tidak, muncul pesan bahwa data tidak ditemukan. Pengguna juga dapat menghapus data kendaraan berdasarkan nomor polisi, di mana kode memanfaatkan deleteFirst, deleteLast, atau deleteAfter sesuai dengan posisi elemen yang akan dihapus. Setelah penghapusan, list yang diperbarui ditampilkan kembali, menunjukkan data kendaraan yang tersisa.

V. KESIMPULAN

Doubly Linked List adalah struktur data yang memungkinkan navigasi dua arah pada elemen-elemen yang terhubung melalui pointer prev dan next, sehingga memberikan fleksibilitas dalam manipulasi data, seperti penambahan dan penghapusan elemen di awal, akhir, atau setelah elemen tertentu dalam list. Dalam implementasi kode, list ini digunakan untuk menyimpan dan mengelola data kendaraan secara dinamis, dengan fungsi-fungsi yang mencakup alokasi memori, pencarian berdasarkan atribut tertentu (misalnya nomor polisi), dan penghapusan data dari berbagai posisi. Pointer First dan Last pada struktur list memastikan akses cepat ke elemen pertama dan terakhir, yang sangat berguna untuk efisiensi dalam operasi penambahan dan penghapusan. Dengan kemampuan untuk melakukan navigasi dua arah, Doubly Linked List menawarkan keunggulan dalam hal efisiensi waktu dan penggunaan memori, terutama pada aplikasi yang membutuhkan pengelolaan data besar secara dinamis. Selain itu, tidak seperti array yang memerlukan penggeseran elemen, Doubly Linked List memungkinkan penghapusan atau penambahan elemen tanpa perlu menggeser data lain, sehingga sangat ideal untuk aplikasi di mana operasi semacam itu sering dilakukan. Dengan kemampuan ini, Doubly Linked List menjadi struktur data yang penting dikuasai, terutama dalam skenario di mana pengelolaan data yang fleksibel dan dinamis sangat dibutuhkan.