LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLY LINKED LIST

Disusun Oleh :

NAMA : AGUNG RAMADHAN

NIM : 103112430060

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Doubly Linked List atau daftar berantai ganda merupakan salah satu jenis struktur data dinamis yang banyak digunakan dalam bidang ilmu komputer. Struktur ini terdiri atas sekumpulan simpul atau *node* yang saling terhubung secara dua arah, yaitu ke simpul sebelumnya melalui pointer prev dan ke simpul berikutnya melalui pointer next. Setiap *node* juga menyimpan elemen data yang menjadi isi dari daftar tersebut. Berbeda dengan *Singly Linked List* yang hanya dapat ditelusuri satu arah, *Doubly Linked List* memungkinkan proses penelusuran ke dua arah sekaligus, baik dari awal (head) menuju akhir (tail), maupun sebaliknya. Keunggulan inilah yang membuat struktur ini efisien dalam operasi pencarian, penyisipan, dan penghapusan elemen di berbagai posisi dalam daftar tanpa perlu menelusuri seluruh data dari awal.

Menurut penelitian yang dilakukan oleh Sundell dan Tsigas (2008) dalam *Journal of Parallel and Distributed Computing*, struktur *Doubly Linked List* memiliki fleksibilitas tinggi dalam manipulasi data, sehingga sering digunakan untuk mendukung implementasi algoritma yang memerlukan akses dua arah dengan efisiensi tinggi, bahkan dalam sistem paralel dan konkuren. Sementara itu, Ng dan Oommen (1992) menjelaskan bahwa *Doubly Linked List* juga dapat dioptimalkan melalui heuristik reorganisasi agar operasi pencarian data menjadi lebih cepat, terutama ketika pola akses data bersifat berulang. Penelitian oleh Khan, Ware, dan Habib (2022) menunjukkan penerapan *Doubly Linked List* pada bidang pengenalan aksi manusia berbasis citra kerangka (*skeleton-based human action recognition*), di mana struktur ini digunakan untuk menyimpan dan memproses urutan data gerakan dengan efisien. Selain itu, Lohmann dan Tutsch (2024) memperkenalkan konsep *Doubly Linked Tree* yang merupakan pengembangan dari *Doubly Linked List* untuk keperluan operasi basis data waktu nyata (*real-time database operations*) pada perangkat keras FPGA.

Secara umum, *Doubly Linked List* memiliki kelebihan berupa kemudahan dalam mengakses data secara dua arah dan kemampuan manipulasi elemen yang lebih fleksibel dibandingkan *Singly Linked List*. Namun, kelemahannya terletak pada penggunaan memori yang lebih besar karena setiap *node* membutuhkan dua pointer tambahan, serta kompleksitas yang lebih tinggi dalam implementasi dan pemeliharaan struktur data. Meskipun demikian, berbagai penelitian menunjukkan bahwa struktur ini tetap relevan dan efektif digunakan dalam banyak aplikasi modern yang memerlukan pengelolaan data dinamis dan efisien, baik dalam konteks perangkat lunak maupun perangkat keras.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

Kode main.cpp

```
// AGUNG RAMADHAN | 103112430060

#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;
    }
    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

```
PS D:\MODUL 6 STRUKDAT> CD .\GUIDED\
PS D:\MODUL 6 STRUKDAT\GUIDED> g++ main.cpp -o main.exe
PS D:\MODUL 6 STRUKDAT\GUIDED> ./main
Awal                    : 5 <-> 10 <-> 20
Setelah delete_first    : 10 <-> 20
Setelah delete_last     : 10 <-> 20
Setelah tambah          : 10 <-> 20 <-> 30 <-> 40
Setelah delete_target   ; 10 <-> 20 <-> 40
PS D:\MODUL 6 STRUKDAT\GUIDED>
```

Deskripsi:

Program C++ di atas mengimplementasikan struktur data doubly linked list (daftar berantai ganda) yang memungkinkan penyimpanan dan pengelolaan data secara dinamis. Setiap simpul (node) memiliki tiga komponen: nilai data, pointer ke node sebelumnya (prev), dan pointer ke node berikutnya (next). Program ini menyediakan berbagai operasi dasar, seperti menambah node di awal (add_first), di akhir (add_last), atau setelah nilai tertentu (add_target), serta menghapus node dari awal (delete_first), dari akhir (delete_last), atau berdasarkan nilai tertentu (delete_target). Selain itu, ada juga fungsi untuk mengubah nilai node (edit_node) dan menampilkan seluruh isi list (view). Pada fungsi main(), program mendemonstrasikan penggunaan fungsi-fungsi tersebut dengan menambah, menghapus, dan menampilkan isi list secara berurutan.

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

Unguided 1

Kode Doublylist.h

```
// AGUNG RAMADHAN | 103112430060

#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H
#include <iostream>
#include <string>
using namespace std;

struct Vehicle {
    string plate;
    string color;
    int year;
};

typedef Vehicle dataType;

struct Node {
    dataType info;
    Node *next;
    Node *prev;
};

typedef Node* pointer;

struct DoubleList {
    pointer head;
    pointer tail;
};

void initList(DoubleList &L);
```

```
pointer createNode(dataType x);
void freeNode(pointer &P);
void showList(DoubleList L);
void addLast(DoubleList &L, pointer P);
pointer searchData(DoubleList L, string plate);
void removeFirst(DoubleList &L, pointer &P);
void removeLast(DoubleList &L, pointer &P);
void removeAfter(pointer prec, pointer &P);

#endif
```

Kode Doublylist.cpp

```
// AGUNG RAMADHAN | 103112430060

#include "Doublylist.h"

void initList(DoubleList &L) {
    L.head = NULL;
    L.tail = NULL;
}

pointer createNode(dataType x) {
    pointer P = new Node;
    P->info = x;
    P->next = NULL;
    P->prev = NULL;
    return P;
}

void freeNode(pointer &P) {
    delete P;
    P = NULL;
}

void addLast(DoubleList &L, pointer P) {
    if (L.head == NULL) {
        L.head = P;
        L.tail = P;
    } else {
        L.tail->next = P;
        P->prev = L.tail;
        L.tail = P;
    }
}

void showList(DoubleList L) {
    pointer P = L.head;
    cout << "=== DAFTAR KENDARAAN ===" << endl;
```

```
    while (P != NULL) {
        cout << "Plat Nomor    : " << P->info.plate << endl;
        cout << "Warna         : " << P->info.color << endl;
        cout << "Tahun Produksi: " << P->info.year << endl;
        cout << endl;
        P = P->next;
    }
}

pointer searchData(DoubleList L, string plate) {
    pointer P = L.head;
    while (P != NULL && P->info.plate != plate) {
        P = P->next;
    }
    return P;
}

void removeFirst(DoubleList &L, pointer &P) {
    if (L.head == NULL)
        P = NULL;
    else if (L.head == L.tail) {
        P = L.head;
        L.head = NULL;
        L.tail = NULL;
    } else {
        P = L.head;
        L.head = L.head->next;
        L.head->prev = NULL;
        P->next = NULL;
    }
}

void removeLast(DoubleList &L, pointer &P) {
    if (L.tail == NULL)
        P = NULL;
    else if (L.head == L.tail) {
        P = L.tail;
        L.head = NULL;
        L.tail = NULL;
    } else {
        P = L.tail;
        L.tail = L.tail->prev;
        L.tail->next = NULL;
        P->prev = NULL;
    }
}

void removeAfter(pointer prec, pointer &P) {
```

```
    if (prec != NULL && prec->next != NULL) {
        P = prec->next;
        prec->next = P->next;
        if (P->next != NULL)
            P->next->prev = prec;
        P->next = NULL;
        P->prev = NULL;
    }
}
```

Kode main.cpp

```
// AGUNG RAMADHAN | 103112430060

#include "Doublylist.h"

int main() {
    DoubleList vehicleList;
    initList(vehicleList);
    dataType inputData;
    pointer P;

    int total;
    cout << "Jumlah Kendaraan: ";
    cin >> total;
    cout << endl;

    for (int i = 0; i < total; i++) {
        cout << "Masukkan Plat Nomor     : ";
        cin >> inputData.plate;
        if (searchData(vehicleList, inputData.plate) != NULL) {
            cout << "Plat Nomor Ini Tidak Ditemukan!" << endl;
            continue;
        }
        cout << "Masukkan Warna Kendaraan: ";
        cin >> inputData.color;
        cout << "Masukkan Tahun Pembuatan: ";
        cin >> inputData.year;
        cout << endl;

        P = createNode(inputData);
        addLast(vehicleList, P);
    }

    showList(vehicleList);

    string searchKey;
```

```
    cout << "Masukkan Plat Nomor Yang Ingin Dicari: ";
    cin >> searchKey;
    pointer found = searchData(vehicleList, searchKey);
    if (found != NULL) {
        cout << "\nData found!" << endl;
        cout << "Plat Nomor    : " << found->info.plate << endl;
        cout << "Warna         : " << found->info.color << endl;
        cout << "Tahun Produksi: " << found->info.year << endl;
    } else {
        cout << "Data Tidak Ditemukan!" << endl;
    }

    cout << "\nMasukkan Plat Nomor Untuk Dihapus: ";
    cin >> searchKey;
    pointer target = searchData(vehicleList, searchKey);
    if (target == NULL) {
        cout << "Data Tidak Ditemukan!" << endl;
    } else {
        if (target == vehicleList.head)
            removeFirst(vehicleList, target);
        else if (target == vehicleList.tail)
            removeLast(vehicleList, target);
        else {
            pointer prec = target->prev;
            removeAfter(prec, target);
        }
        freeNode(target);
        cout << "Data Berhasil Dihapus!\n";
    }

    cout << "\nData Diperbarui:\n";
    showList(vehicleList);

    return 0;
}
```

Screenshots Output

```
PS D:\MODUL 6 STRUKDAT\UNGUIDED> g++ main.cpp doublylist.cpp -o main.exe
PS D:\MODUL 6 STRUKDAT\UNGUIDED> ./main
Jumlah Kendaraan: 3

Masukkan Plat Nomor     : B4332BAF
Masukkan Warna Kendaraan: PUTIH
Masukkan Tahun Pembuatan: 2016

Masukkan Plat Nomor     : B4546BRE
Masukkan Warna Kendaraan: HITAM
Masukkan Tahun Pembuatan: 2025

Masukkan Plat Nomor     : B4748BYU
Masukkan Warna Kendaraan: HITAM
Masukkan Tahun Pembuatan: 2021

=== DAFTAR KENDARAAN ===
Plat Nomor    : B4332BAF
Warna         : PUTIH
Tahun Produksi: 2016

Plat Nomor    : B4546BRE
Warna         : HITAM
Tahun Produksi: 2025

Plat Nomor    : B4748BYU
Warna         : HITAM
Tahun Produksi: 2021
```

```
Masukkan Plat Nomor Yang Ingin Dicari: B4546BRE

Data found!
Plat Nomor    : B4546BRE
Warna         : HITAM
Tahun Produksi: 2025

Masukkan Plat Nomor Untuk Dihapus: B4748BYU
Data Berhasil Dihapus!

Data Diperbarui:
=== DAFTAR KENDARAAN ===
Plat Nomor    : B4332BAF
Warna         : PUTIH
Tahun Produksi: 2016

Plat Nomor    : B4546BRE
Warna         : HITAM
Tahun Produksi: 2025
```

Deskripsi:

Kode program ini merupakan implementasi struktur data Doubly Linked List menggunakan bahasa C++ yang dibagi menjadi tiga file utama, yaitu Doublylist.h, Doublylist.cpp, dan main.cpp. File Doublylist.h berisi deklarasi struktur data dan fungsi. Di dalamnya terdapat struktur Vehicle untuk menyimpan data kendaraan berupa plat, warna, dan tahun, serta struktur Node yang memiliki pointer next dan prev sebagai penghubung antar simpul. Struktur DoubleList digunakan untuk menandai awal dan akhir list melalui pointer head dan tail.

File Doublylist.cpp berisi implementasi fungsi-fungsi dasar seperti initList() untuk inisialisasi list, createNode() untuk membuat simpul baru, dan addLast() untuk menambahkan data di akhir list. Fungsi showList() digunakan untuk menampilkan isi list, sedangkan searchData() untuk mencari data berdasarkan plat kendaraan. Program juga menyediakan fungsi penghapusan, seperti removeFirst(), removeLast(), dan removeAfter() untuk menghapus simpul sesuai posisinya, serta freeNode() untuk membebaskan memori yang tidak terpakai.

Sementara itu, file main.cpp berfungsi sebagai program utama yang mengatur alur logika program. Pengguna dapat menambahkan data kendaraan, menampilkan seluruh data, mencari kendaraan berdasarkan plat, dan menghapus data tertentu dari list. Setelah operasi dilakukan, program akan menampilkan kondisi list terbaru. Secara keseluruhan, kode ini memperlihatkan penerapan konsep dasar *Doubly Linked List* dengan operasi penting seperti penambahan, pencarian, penghapusan, dan penelusuran data secara dua arah.

D. Kesimpulan

Kesimpulannya, kode program ini menunjukkan bagaimana struktur data Doubly Linked List dapat diimplementasikan secara efektif menggunakan bahasa C++. Melalui pembagian kode ke dalam tiga file utama (Doublylist.h, Doublylist.cpp, dan main.cpp), program ini menerapkan prinsip modularitas yang memudahkan pemeliharaan serta pengembangan kode. Program mampu menjalankan operasi dasar yang umum pada *linked list*, seperti menambah data, menampilkan isi list, mencari data berdasarkan kunci tertentu, serta menghapus node baik di awal, akhir, maupun tengah list.

Struktur *Doubly Linked List* yang digunakan memungkinkan proses penelusuran data dilakukan dua arah, sehingga memberikan fleksibilitas dan efisiensi dalam manipulasi data dibandingkan *Singly Linked List*. Selain itu, penggunaan fungsi dinamis seperti createNode() dan freeNode() memperlihatkan bagaimana pengelolaan memori dilakukan dengan baik untuk mencegah kebocoran memori. Secara keseluruhan, program ini tidak hanya memperlihatkan konsep dasar dari *Doubly Linked List*, tetapi juga mengajarkan praktik pemrograman terstruktur yang penting dalam membangun aplikasi berbasis struktur data dinamis.

E. Referensi

Sundell, H., & Tsigas, P. (2008). *Lock-Free Deques and Doubly Linked Lists*. *Journal of Parallel and Distributed Computing*, 68(7), 1008-1020.

Ng, D. T. H., & Oommen, B. J. (1992). *A Short Note on Doubly-Linked List Reorganizing Heuristics*. *The Computer Journal*, 35(5), 533-535.

Khan, M. S., Ware, A., Habib, U., Khalid, M. J., & Bahoo, N. (2022). *Skeleton Based Human Action Recognition Using Doubly Linked List*. *International Journal of Computer Trends and Technology*, 70(2), 18-21.

Lohmann, S., & Tutsch, D. (2024). *The Doubly Linked Tree of Singly Linked Rings: Providing Hard Real-Time Database Operations on an FPGA*. *Computers*, 13(1), 8.