LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL V
PENGENALAN CODE BLOCKS

Disusun Oleh :
NAMA : Sinta Sintiani
NIM : 103112430047

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Doubly Linked List adalah struktur data dinamis yang terdiri atas sekumpulan simpul (*node*), di mana setiap simpul memiliki tiga komponen utama, yaitu pointer ke simpul sebelumnya (*prev*), data yang disimpan (*data*), dan pointer ke simpul berikutnya (*next*). Berbeda dengan *singly linked list* yang hanya dapat ditelusuri satu arah, *doubly linked list* memungkinkan penelusuran dua arah, baik maju maupun mundur. Hal ini memudahkan proses penyisipan dan penghapusan data di berbagai posisi dalam daftar tanpa harus menelusuri dari awal. Meskipun demikian, struktur ini membutuhkan memori lebih besar karena setiap simpul menyimpan dua pointer, serta memerlukan pengelolaan pointer yang lebih kompleks agar tidak terjadi kesalahan dalam operasi manipulasi data.

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;
        }
        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();

    return 0;
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA> cd "d:\SEM 3\STRUKTUR DATA\mod 5\" ; if ($?) {
Awal                    : 5 <-> 10 <-> 20
Setelah delete_first    : 10 <-> 20
Setelah delete_last     : 10
Setelah tambah          : 10 <-> 30 <-> 40
Setelah delete_target   : 10 <-> 40
```

Deskripsi:

Program di atas merupakan implementasi Doubly Linked List dalam C++ yang mendukung operasi penambahan, penghapusan, penelusuran, dan pengeditan data. Setiap node memiliki tiga elemen, yaitu data, prev, dan next. Fungsi-fungsi seperti add_first(), add_last(), delete_first(), delete_last(), dan delete_target() digunakan untuk memanipulasi data pada berbagai posisi, sedangkan view() menampilkan isi list. Melalui fungsi main(), program menampilkan cara kerja *doubly linked list* yang dapat diakses dua

arah secara efisien.

Unguided 1

Doublylist.h

```h
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <iostream>
#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;
};

void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertLast(List &L, address P);
address findElm(List L, string nopol);
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
void deleteByNopol(List &L, string nopol);

#endif
```

Doublylist.cpp

```cpp
#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 printInfo(List L) {
    cout << "DATA LIST 1" << endl;
    cout << "-------------------" << endl;

    address P = L.First;
    while (P != NULL) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna        : " << P->info.warna << endl;
        cout << "Tahun        : " << P->info.thnBuat << endl;
        cout << "-------------------" << endl;
        P = P->next;
    }
}

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 findElm(List L, string nopol) {
    address P = L.First;
    while (P != NULL) {
```

```
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return NULL;
}

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 = L.First->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 = L.Last->prev;
            L.Last->next = NULL;
            P->prev = NULL;
        }
    }
}

void deleteAfter(address Prec, address &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;
    }
}
```

```
void deleteByNopol(List &L, string nopol) {
    address P = findElm(L, nopol);

    if (P == NULL) {
        cout << "Data dengan nomor polisi " << nopol << " tidak
ditemukan." << endl;
        return;
    }

    if (P == L.First) {
        deleteFirst(L, P);
    } else if (P == L.Last) {
        deleteLast(L, P);
    } else {
        deleteAfter(P->prev, P);
    }

    dealokasi(P);
    cout << "Data dengan nomor polisi " << nopol << " berhasil dihapus."
<< endl;
}
```

Main.cpp

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

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

    infotype kendaraan;
    string nopolCari, nopolHapus;
    address P;

    cout << "Masukkan nomor polisi: ";
    cin >> kendaraan.nopol;
    cout << "Masukkan warna kendaraan: ";
    cin >> kendaraan.warna;
    cout << "Masukkan tahun kendaraan: ";
    cin >> kendaraan.thnBuat;

    P = alokasi(kendaraan);
    if (findElm(L, kendaraan.nopol) == NULL) {
```

```
        insertLast(L, P);
    } else {
        cout << "Nomor polisi sudah terdaftar" << endl;
        dealokasi(P);
    }

    cout << "\nMasukkan nomor polisi: ";
    cin >> kendaraan.nopol;
    cout << "Masukkan warna kendaraan: ";
    cin >> kendaraan.warna;
    cout << "Masukkan tahun kendaraan: ";
    cin >> kendaraan.thnBuat;

    P = alokasi(kendaraan);
    if (findElm(L, kendaraan.nopol) == NULL) {
        insertLast(L, P);
    } else {
        cout << "Nomor polisi sudah terdaftar" << endl;
        dealokasi(P);
    }

    cout << "\nMasukkan nomor polisi: ";
    cin >> kendaraan.nopol;
    cout << "Masukkan warna kendaraan: ";
    cin >> kendaraan.warna;
    cout << "Masukkan tahun kendaraan: ";
    cin >> kendaraan.thnBuat;

    P = alokasi(kendaraan);
    if (findElm(L, kendaraan.nopol) == NULL) {
        insertLast(L, P);
    } else {
        cout << "Nomor polisi sudah terdaftar" << endl;
        dealokasi(P);
    }

    cout << "\nMasukkan nomor polisi: ";
    cin >> kendaraan.nopol;
    cout << "Masukkan warna kendaraan: ";
    cin >> kendaraan.warna;
    cout << "Masukkan tahun kendaraan: ";
    cin >> kendaraan.thnBuat;

    P = alokasi(kendaraan);
    if (findElm(L, kendaraan.nopol) == NULL) {
        insertLast(L, P);
    } else {
        cout << "Nomor polisi sudah terdaftar" << endl;
```

```
        dealokasi(P);
    }

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

    cout << "\nMasukkan Nomor Polisi yang dicari: ";
    cin >> nopolCari;

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

    cout << "\nMasukkan Nomor Polisi yang akan dihapus: ";
    cin >> nopolHapus;

    deleteByNopol(L, nopolHapus);

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

    return 0;
}
```

Screenshots Output

```
PS D:\SEM 3\STRUKTUR DATA\Modul 5> cd "d:\SEM 3\STRUKTUR DATA\Modul 5\" ; if ($?)
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

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


DATA LIST 1
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

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

DATA LIST 1
-------------------
Nomor Polisi : D001
Warna        : hitam
Tahun        : 90
-------------------
Nomor Polisi : D004
Warna        : kuning
Tahun        : 90
-------------------
```

Deskripsi:

C. Melalui praktikum ini, dapat disimpulkan bahwa Doubly Linked List merupakan struktur data dinamis yang setiap elemennya memiliki dua pointer, yaitu next untuk menunjuk ke elemen berikutnya dan prev untuk menunjuk ke elemen sebelumnya. Dengan adanya dua arah penunjuk ini, proses penelusuran data, penyisipan (insert), serta penghapusan (delete) menjadi lebih fleksibel dibandingkan *Singly Linked List*, karena dapat dilakukan dari dua arah (maju dan mundur).Kesimpulan

D. Referensi

https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html

https://rumahcoding.co.id/linked-list-pengertian-dan-implementasi-dasar/