LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
PENGENALAN DOUBLY LINKED LIST

Disusun Oleh :
NAMA : Azzahra Farelika Esti Ning Tyas
NIM : 103112430023

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Doubly Linked List merupakan salah satu bentuk struktur data dinamis yang digunakan untuk menyimpan sekumpulan elemen yang saling terhubung melalui pointer. Setiap elemen dalam Doubly Linked List disebut node dan memiliki tiga komponen utama, yaitu: data (info), pointer ke elemen sebelumnya (prev), dan pointer ke elemen berikutnya (next). Berbeda dengan Singly Linked List yang hanya memiliki satu arah hubungan antar-node (maju), Doubly Linked List memungkinkan traversal dua arah baik maju maupun mundu r karena adanya pointer ganda tersebut. Hal ini menjadikan operasi seperti penyisipan (insert), penghapusan (delete), maupun pencarian (searching) elemen menjadi lebih fleksibel dan efisien dalam konteks tertentu.

Dalam implementasi Doubly Linked List, terdapat dua pointer utama, yaitu first yang menunjuk ke elemen pertama dan last yang menunjuk ke elemen terakhir dari list. Operasi dasar yang umum dilakukan antara lain adalah insertFirst, insertLast, insertAfter, deleteFirst, deleteLast, dan deleteAfter. Setiap operasi tersebut mengubah hubungan antar-node dengan memperhatikan keseimbangan antara pointer prev dan next agar struktur list tetap konsisten. Kelebihan utama Doubly Linked List terletak pada kemampuannya untuk melakukan iterasi dua arah dan efisiensi dalam penghapusan atau penyisipan elemen di posisi tertentu tanpa perlu traversal dari awal list. Namun, struktur ini juga memerlukan memori lebih besar karena setiap node menyimpan dua pointer sekaligus.

Dalam pemrograman, implementasi Doubly Linked List biasanya menggunakan bahasa seperti C atau C++, di mana setiap node direpresentasikan melalui struct yang berisi field info, next, dan prev. Konsep ini sangat penting dalam pembelajaran struktur data karena menjadi dasar bagi implementasi struktur yang lebih kompleks seperti deque, navigation history, serta memory management systems.

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

Guided 1

main.cpp

Modul06 > GUIDED > main.cpp > add_target(int, int)

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

Windsurf: Refactor | Explain
struct Node
{
    int data;
    Node* prev;
    Node* next;
};

Node*ptr_first = NULL;
Node*ptr_last = NULL;

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;
}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;
}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;
        }
    }
}
```

main.cpp

Modul06 > GUIDED > main.cpp > add_target(int, int)

```cpp
void add_target(int targetValue, int newValue)
{

}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;

}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;
}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
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;
}

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment | ×
void delete_target(int targetValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
```

```cpp
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:\StrukturData> cd "d:\StrukturData\Modul06\GUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
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 bahasa C++. Struktur data ini menggunakan node yang memiliki tiga bagian: data, prev, dan next, yang memungkinkan penelusuran dua arah. Pointer global ptr_first dan ptr_last digunakan untuk menandai elemen pertama dan terakhir dalam list. Fungsi add_first dan add_last menambah elemen di awal atau akhir list, sedangkan add_target menyisipkan elemen baru setelah nilai tertentu. Fungsi delete_first, delete_last, dan delete_target digunakan untuk menghapus elemen dari posisi berbeda. Fungsi edit_node berfungsi mengubah nilai

suatu elemen, dan view menampilkan isi list secara berurutan. Melalui fungsi main, seluruh operasi tersebut diuji untuk menunjukkan proses penambahan, penghapusan, serta penelusuran data dalam struktur Doubly Linked List.

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

Unguided 1

doublylist.cpp

Modul06 > UNGUIDED > doublylist.cpp > printInfo(List)

```cpp
#include "doublylist.h"

// Membuat list kosong
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void CreateList(List &L) {
    L.First = NULL;
    L.Last = NULL;
}

// Mengalokasikan memori untuk elemen baru
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
address alokasi(infotype X) {
    address P = new ElmList;
    P->info = X;
    P->next = NULL;
    P->prev = NULL;
    return P;
}

// Dealokasi memori
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Menampilkan semua informasi dalam list
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void printInfo(List L) {
    if (L.First == NULL) {
        cout << "List kosong" << endl;
        return;
    }

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

// Menyisipkan elemen di akhir list
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
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;
    }
}

// Mencari elemen berdasarkan nomor polisi
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
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;
}

// Mengecek apakah nomor polisi sudah ada
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
bool isNopolExist(List L, string nopol) {
    return findElm(L, nopol) != NULL;
}

// Menghapus elemen pertama
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void deleteFirst(List &L, address &P) {
    if (L.First == NULL) {
        P = NULL;
        return;
    }

    P = L.First;
    if (L.First == L.Last) {
```

doublylist.cpp

Modul06 > UNGUIDED > doublylist.cpp > printInfo(List)

```cpp
bool isNopolExist(List L, string nopol) {
}

// Menghapus elemen pertama
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void deleteFirst(List &L, address &P) {
    if (L.First == NULL) {
        P = NULL;
        return;
    }

    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;
    P->prev = NULL;
}

// Menghapus elemen terakhir
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void deleteLast(List &L, address &P) {
    if (L.Last == NULL) {
        P = NULL;
        return;
    }

    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;
}

// Menghapus elemen setelah Prec
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void deleteAfter(address Prec, address &P) {
    if (Prec == NULL || Prec->next == NULL) {
        P = NULL;
        return;
    }

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

    P->next = NULL;
    P->prev = NULL;
}

// Menghapus berdasarkan nomor polisi
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
bool deleteByNopol(List &L, string nopol) {
    address P = findElm(L, nopol);
    if (P == NULL) {
        return false; // Data tidak ditemukan
    }

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

    return true;
}
```

doublylist.h

Modul06 > UNGUIDED > doublylist.h > ...

```h
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <iostream>
#include <string>
using namespace std;

Windsurf: Refactor | Explain
struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};

typedef kendaraan infotype;
typedef struct ElmList *address;

Windsurf: Refactor | Explain
struct ElmList {
    infotype info;
    address next;
    address prev;
};

Windsurf: Refactor | Explain
struct List {
    address First;
    address Last;
};

// Prosedur dan Fungsi
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
void CreateList(List &L);
Tabnine | Edit | Test | Explain | Document
address alokasi(infotype x);
Tabnine | Edit | Test | Explain | Document
void dealokasi(address &P);
Tabnine | Edit | Test | Explain | Document
void printInfo(List L);
Tabnine | Edit | Test | Explain | Document
void insertLast(List &L, address P);

// 🔽 Tambahkan deklarasi berikut
Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain |
address findElm(List L, string nopol);
Tabnine | Edit | Test | Explain | Document
bool isNopolExist(List L, string nopol);
Tabnine | Edit | Test | Explain | Document
void deleteFirst(List &L, address &P);
Tabnine | Edit | Test | Explain | Document
void deleteLast(List &L, address &P);
Tabnine | Edit | Test | Explain | Document
void deleteAfter(address Prec, address &P);
Tabnine | Edit | Test | Explain | Document
bool deleteByNopol(List &L, string nopol);

#endif
```

main.cpp

Modul06 > UNGUIDED > main.cpp > main()

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

Tabnine | Edit | Test | Explain | Document | Windsurf: Refactor | Explain | Generate Function Comment |
int main() {
    List L;
    CreateList(L);

    int jumlahData = 0;
    const int MAX_DATA = 3;

    while (jumlahData < MAX_DATA) {
        infotype kendaraan;

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

        if (isNopolExist(L, kendaraan.nopol)) {
            cout << "Nomor polisi sudah terdaftar!" << endl << endl;
        } else {
            address P = alokasi(kendaraan);
            insertLast(L, P);
            jumlahData++;
        }
    }

    cout << "=== Data kendaraan yang terdaftar ===" << endl;
    printInfo(L);

    // Mencari data berdasarkan nomor polisi
    string nopolCari;
    cout << "Masukkan nomor polisi yang ingin dicari: ";
    cin >> nopolCari;

    address hasilCari = findElm(L, nopolCari);
    if (hasilCari != NULL) {
        cout << endl << "Data ditemukan:" << endl;
        cout << "nopol : " << hasilCari->info.nopol << endl;
        cout << "warna : " << hasilCari->info.warna << endl;
        cout << "tahun : " << hasilCari->info.thnBuat << endl;
    } else {
        cout << "Data dengan nopol tersebut tidak ditemukan." << endl;
    }

    cout << endl;


    // Menghapus data berdasarkan nomor polisi
    string nopolHapus;
    cout << "Masukkan nomor polisi yang ingin dihapus: ";
    cin >> nopolHapus;

    if (deleteByNopol(L, nopolHapus)) {
        cout << "Data berhasil dihapus." << endl;
    } else {
        cout << "Data tidak ditemukan." << endl;
    }

    cout << endl << "=== Data setelah penghapusan ===" << endl;
    printInfo(L);

    return 0;
}
```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul06\UNGUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
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 kendaraan yang terdaftar ===
nopol : D001
warna : hitam
tahun : 90

nopol : D003
warna : putih
tahun : 70

nopol : D004
warna : kuning
tahun : 90

Masukkan nomor polisi yang ingin dicari: D001

Data ditemukan:
nopol : D001
warna : hitam
tahun : 90

Masukkan nomor polisi yang ingin dihapus: D003
Data berhasil dihapus.

=== Data setelah penghapusan ===
nopol : D001
warna : hitam
tahun : 90

nopol : D004
warna : kuning
tahun : 90
```

Deskripsi:

Program di atas merupakan implementasi struktur data Doubly Linked List untuk menyimpan dan mengelola data kendaraan menggunakan bahasa C++. Setiap elemen list menyimpan informasi berupa nomor polisi, warna, dan tahun pembuatan kendaraan. File doublylist.h mendefinisikan struktur data kendaraan, tipe ElmList sebagai node yang memiliki pointer next dan prev, serta deklarasi fungsi-fungsi yang digunakan. File doublylist.cpp mengimplementasikan berbagai operasi dasar Doubly Linked List, seperti membuat list kosong (CreateList), menambahkan data di akhir (insertLast), menampilkan seluruh isi list (printInfo), mencari data berdasarkan nomor polisi (findElm), memeriksa duplikasi data (isNopolExist), dan menghapus elemen berdasarkan posisi atau nomor polisi (deleteFirst, deleteLast, deleteAfter, deleteByNopol).

Dalam main.cpp, program meminta pengguna untuk memasukkan maksimal tiga data kendaraan, kemudian memeriksa apakah nomor polisi sudah terdaftar sebelum disimpan ke dalam list. Setelah semua data dimasukkan, program menampilkan daftar kendaraan

yang tersimpan, memungkinkan pengguna untuk mencari data berdasarkan nomor polisi, serta menghapus data tertentu menggunakan fungsi deleteByNopol. Setelah penghapusan, list yang tersisa ditampilkan kembali. Secara keseluruhan, program ini menunjukkan penerapan konsep Doubly Linked List yang mencakup proses penambahan, pencarian, dan penghapusan data dengan menjaga hubungan dua arah antar elemen agar struktur list tetap konsisten.

D. Kesimpulan

Dari praktikum Modul VI mengenai Doubly Linked List, dapat disimpulkan bahwa struktur data ini merupakan bentuk pengembangan dari Singly Linked List yang memungkinkan proses penelusuran dua arah, baik maju maupun mundur. Setiap node memiliki dua pointer, yaitu prev dan next, yang membuat proses penyisipan, penghapusan, serta pencarian data menjadi lebih fleksibel dan efisien. Implementasi program menggunakan bahasa C++ menunjukkan bagaimana hubungan antar-node diatur agar list tetap konsisten, terutama saat terjadi penambahan atau penghapusan data. Melalui latihan guided dan unguided, mahasiswa dapat memahami prinsip dasar kerja Doubly Linked List serta mengaplikasikannya dalam kasus nyata seperti pengelolaan data kendaraan.

E. Referensi

Modul Praktikum Struktur Data, Telkom University Purwokerto. Modul VI – Pengenalan Doubly Linked List (2025).

Munir, R. (2019). Algoritma dan Struktur Data. Bandung: Informatika.

Koffman, E. B., & Wolfgang, P. A. T. (2010). Data Structures: Abstraction and Design Using Java. John Wiley & Sons.

Wibowo, A. (2016). Struktur Data dengan C++. Yogyakarta: Andi.