

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL IV DAN VI
DOUBLY LINKED LIST**



Disusun Oleh :

NAMA : Balawan Satria Lhaksana Putra M.

NIM : 103112430004

Dosen

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Doubly Linked List atau *Linked List Ganda* merupakan salah satu jenis struktur data dinamis yang terdiri atas sekumpulan node, di mana setiap node memiliki tiga komponen utama yaitu data, pointer ke node sebelumnya (*prev*), dan pointer ke node berikutnya (*next*). Berbeda dengan *Singly Linked List* yang hanya dapat ditelusuri dalam satu arah, Doubly Linked List memungkinkan penelusuran data baik dari depan maupun dari belakang karena setiap node saling terhubung dua arah. Keunggulan dari Doubly Linked List terletak pada fleksibilitasnya dalam melakukan operasi penambahan dan penghapusan elemen di posisi mana pun tanpa perlu menelusuri list dari awal, sementara kelemahannya adalah penggunaan memori yang lebih besar karena setiap node harus menyimpan dua pointer. Dalam implementasinya menggunakan bahasa C++, struktur ini didefinisikan dengan struct yang berisi atribut data serta pointer prev dan next. Operasi dasar yang umum dilakukan meliputi pembuatan list baru dengan fungsi `CreateList()`, penambahan elemen di akhir list menggunakan `InsertLast()`, pencarian elemen dengan `FindElm()`, penghapusan node melalui `DeleteFirst()`, `DeleteLast()`, dan `DeleteAfter()`, serta penampilan seluruh isi list menggunakan `PrintInfo()`. Doubly Linked List sering digunakan dalam berbagai aplikasi seperti sistem navigasi *browser history*, daftar putar musik (*music playlist*), dan fitur *undo-redo* pada perangkat lunak karena kemampuannya menelusuri dan memanipulasi data secara dua arah dengan efisien.

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

Guided 1 (main.cpp)

```
#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;
        }
        else if (current == ptr_last)
        {
            delete_last();
            return;
        }
        else
        {
            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();

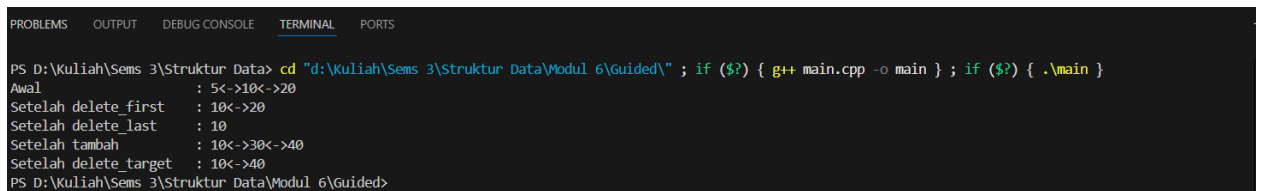
delete_last();
cout << "Setelah delete_last\t: ";
view();

add_last(30);
add_last(40);
cout << "Setelah tambah\t\t: ";
view();

delete_target(30);
cout << "Setelah delete_target\t: ";
view();
}

```

Screenshots Output



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\Kuliah\Sems 3\Struktur Data> cd "d:\Kuliah\Sems 3\Struktur Data\Modul 6\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
PS D:\Kuliah\Sems 3\Struktur Data\Modul 6\Guided>

```

Deskripsi:

Program ini merupakan implementasi dari struktur data Doubly Linked List dalam bahasa C++, di mana setiap elemen (node) memiliki dua pointer yaitu `prev` yang menunjuk ke node sebelumnya dan `next` yang menunjuk ke node berikutnya. Setiap node menyimpan data berupa bilangan bulat (`int`) dan diatur menggunakan dua pointer utama, `ptr_first` sebagai penanda node pertama dan `ptr_last` sebagai penanda node terakhir. Program ini memiliki beberapa fungsi utama, seperti menambah node di awal (`add_first`), di akhir (`add_last`), atau setelah nilai tertentu (`add_target`), serta menghapus node pertama (`delete_first`), node terakhir (`delete_last`), atau node dengan nilai tertentu (`delete_target`). Selain itu, terdapat juga fungsi untuk mengedit nilai data (`edit_node`) dan menampilkan seluruh isi list (`view`). Melalui fungsi `main`, program menampilkan proses penambahan, penghapusan, dan penelusuran data secara berurutan, sehingga menggambarkan cara kerja Doubly Linked List dalam mengelola data secara dinamis menggunakan pointer di C++.

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

Unguided 1 (Doublylist.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;

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

typedef ElmList* address;

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, infotype x);

void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);

#endif
```

Unguided 1 (Doublylist.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;
    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, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info.nopol == x.nopol) {
            return P;
        }
        P = P->next;
    }
}

```



```

        return NULL;
    }

void deleteFirst(List &L, address &P) {
    if (L.First == NULL) {
        P = NULL;
    } else if (L.First == L.Last) {
        P = L.First;
        L.First = NULL;
        L.Last = NULL;
    } else {
        P = L.First;
        L.First = P->next;
        L.First->prev = NULL;
        P->next = NULL;
    }
}

void deleteLast(List &L, address &P) {
    if (L.Last == NULL) {
        P = NULL;
    } else if (L.First == L.Last) {
        P = L.Last;
        L.First = NULL;
        L.Last = NULL;
    } else {
        P = L.Last;
        L.Last = P->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;
    } else {
        P = NULL;
    }
}

```

```

#include "Doublylist.h"
#include "Doublylist.cpp"
#include <iostream>

using namespace std;

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

    infotype x;
    address P;

    for (int i = 0; i < 4; i++) {
        cout << "Masukkan Nomor Polisi : ";
        cin >> x.nopol;

        infotype temp;
        temp.nopol = x.nopol;

        if (findElm(L, temp) != NULL) {
            cout << "Nomor polisi sudah terdaftar\n\n";
            continue;
        }

        cout << "Masukkan Warna Kendaraan : ";
        cin >> x.warna;
        cout << "Masukkan Tahun Kendaraan : ";
        cin >> x.thnBuat;
        cout << endl;

        P = alokasi(x);
        insertLast(L, P);
    }

    cout << endl;
    printInfo(L);

    infotype cari;
    cout << "Masukkan Nomor Polisi yang dicari : ";
    cin >> cari.nopol;

    address found = findElm(L, cari);
    if (found != NULL) {
        cout << 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;
    }

    cout << endl;

    infotype hapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus : ";
    cin >> hapus.nopol;

    address del = findElm(L, hapus);
    if (del != NULL) {
        if (del == L.First) {
            deleteFirst(L, del);
        } else if (del == L.Last) {
            deleteLast(L, del);
        } else {
            deleteAfter(del->prev, del);
        }
        cout << "Data dengan nomor polisi " << hapus.nopol << " berhasil
dihapus." << endl;
        dealokasi(del);
    } else {
        cout << "Data tidak ditemukan." << endl;
    }

    cout << endl;
    printInfo(L);

    return 0;
}

```

Screenshots Output

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

DATA LIST 1
Nomor Polisi : D001
Warna : hitam
Tahun : 90

Nomor Polisi : D003
Warna : putih
Tahun : 70

Nomor Polisi : D002
Warna : merah
Tahun : 90

Masukkan Nomor Polisi yang dicari : D003

Nomor Polisi : D003
Warna : putih
Tahun : 70

Masukkan Nomor Polisi yang akan dihapus : D002
Data dengan nomor polisi D002 berhasil dihapus.

DATA LIST 1
Nomor Polisi : D001
Warna : hitam
Tahun : 90

Nomor Polisi : D003
Warna : putih
Tahun : 70

PS D:\Kuliah\Sems 3\Struktur Data\Modul 6\Unguided> |
```

Deskripsi:

Program ini merupakan implementasi dari struktur data Doubly Linked List dalam bahasa C++ yang digunakan untuk mengelola data kendaraan. Setiap node dalam list menyimpan informasi berupa nomor polisi, warna kendaraan, dan tahun pembuatan, serta dua pointer yaitu `next` untuk menunjuk ke elemen berikutnya dan `prev` untuk menunjuk ke elemen sebelumnya. Program dibagi menjadi tiga bagian, yaitu file header `Doublylist.h`, file implementasi `Doublylist.cpp`, dan file utama `main.cpp`. Fungsi-fungsi yang digunakan meliputi pembuatan list baru (`CreateList`), penambahan data di akhir list (`insertLast`), pencarian data berdasarkan nomor polisi (`findElm`), penghapusan data di posisi awal, akhir, atau tengah (`deleteFirst`, `deleteLast`, `deleteAfter`), serta penampilan seluruh data kendaraan (`printInfo`). Dalam prosesnya, program meminta pengguna untuk memasukkan data kendaraan, memastikan tidak ada nomor polisi yang duplikat, menampilkan seluruh data, mencari kendaraan tertentu, dan menghapus data berdasarkan nomor polisi. Melalui implementasi ini, pengguna dapat memahami cara kerja Doubly Linked List dalam melakukan operasi tambah, hapus, dan cari data secara dinamis menggunakan pointer di C++.

D. Kesimpulan

Dari percobaan yang dilakukan, dapat disimpulkan bahwa Doubly Linked List merupakan struktur data yang efisien untuk pengelolaan data secara dinamis. Dengan adanya dua pointer dalam setiap node, proses penambahan, penghapusan, dan pencarian data dapat dilakukan lebih fleksibel dibandingkan dengan Singly Linked List. Program yang dibuat mampu menampilkan data kendaraan, menambah data baru, mencari data berdasarkan nomor polisi, serta menghapus data tertentu tanpa perlu menggeser elemen lain secara manual. Implementasi ini memperlihatkan bagaimana konsep pointer dan relasi dua arah dapat dimanfaatkan untuk mengelola data dalam struktur list secara optimal menggunakan bahasa C++.

E. Referensi

- https://www.w3schools.com/dsa/dsa_data_linkedlists_types.php
- <https://www.programiz.com/dsa/doubly-linked-list>
- https://en.wikipedia.org/wiki/Doubly_linked_list