

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL VI VI
SINGLY LINKED LIST**



Disusun Oleh :

NAMA : RISKY CAHAYU

NIM : 103112430121

Dosen

FAHRUDIN MUKTI WIBOWO

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

A. Dasar Teori

Single Linked List merupakan salah satu struktur data dinamis yang terdiri atas rangkaian elemen yang disebut node, di mana setiap node menyimpan dua bagian utama yaitu data dan pointer (next) yang menunjuk ke node berikutnya. Struktur ini memungkinkan penyimpanan dan pengelolaan data secara efisien karena ukurannya dapat berubah secara dinamis selama program berjalan, berbeda dengan array yang memiliki ukuran tetap. Operasi dasar pada Single Linked List meliputi penambahan data di awal, tengah, atau akhir daftar, penghapusan data, serta penelusuran seluruh elemen dari node pertama (head) hingga node terakhir (tail) yang menunjuk ke NULL. Struktur ini banyak digunakan dalam berbagai aplikasi seperti sistem playlist lagu, antrian, dan implementasi stack, karena memungkinkan penyisipan dan penghapusan data tanpa perlu menggeser elemen lain. Meskipun demikian, Single Linked List memiliki kelemahan yaitu tidak dapat mengakses elemen secara langsung dan membutuhkan memori tambahan untuk penyimpanan pointer. (Sumber: Wikipedia, "Linked List," 2024; Knuth, D.E., The Art of Computer Programming, 1998).

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

Guided 1

Kode main.cpp

```
// Risky Cahayu
// 103112430121

#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();

return 0;
}

```

Screenshots Output

```

PS C:\Users\ASUS\Videos\strukdat\Laprak Modul 4> cd "c:\Users\ASUS\Videos\strukdat\Modul4_5.cpp\" ; if ($?) { g++ main.cpp -
o main } ; if ($?) { .\main }
Mengisi list menggunakan insertLast...
Isi list sekarang adalah; 9 12 8 0 2
Press any key to continue . . .
PS C:\Users\ASUS\Videos\strukdat\Modul4_5.cpp>

```

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

```
// Risky Cahayu
// 103112430121

#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;
};

// PROTOTYPE
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);

#endif
```

Kode Doublylist.cpp

```
// Risky Cahayu
// 103112430121

#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 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;
    }
}

void printInfo(List L) {
    address P = L.first;
    cout << "DATA LIST :" << endl;
    while (P != NULL) {
        cout << "No Polisi : " << P->info.nopol << endl;
        cout << "Warna      : " << P->info.warna << endl;
        cout << "Tahun      : " << P->info.thnBuat << endl;
        cout << endl;
        P = P->next;
    }
}

address findElm(List L, string nopol) {
```

```

        address P = L.first;
        while (P != NULL && P->info.nopol != nopol) {
            P = P->next;
        }
        return P;
    }

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 = L.first->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 = 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;
    }
}

```

Kode main.cpp

```
// Risky Cahayu
// 103112430121

#include "Doublylist.h"

int main() {
    List L;
    createList(L);
    infotype x;
    address P;

    int n;
    cout << "Masukkan jumlah kendaraan: ";
    cin >> n;
    cout << endl;

    for (int i = 0; i < n; i++) {
        cout << "Masukkan nomor polisi: ";
        cin >> x.nopol;
        if (findElm(L, x.nopol) != NULL) {
            cout << "Nomor polisi sudah terdaftar!" << endl;
            continue;
        }
        cout << "Masukkan warna kendaraan: ";
        cin >> x.warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> x.thnBuat;
        cout << endl;

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

    printInfo(L);

    // Cari data
    string cari;
    cout << "Masukkan nomor polisi yang dicari: ";
    cin >> cari;
    address found = findElm(L, cari);
    if (found != NULL) {
        cout << "Ditemukan!" << endl;
        cout << "No Polisi : " << found->info.nopol << endl;
        cout << "Warna      : " << found->info.warna << endl;
        cout << "Tahun      : " << found->info.thnBuat << endl;
    } else {
        cout << "Data tidak ditemukan!" << endl;
    }
}
```

```

    }

    // Hapus data dengan nomor polisi D003
    cout << "\nMasukkan nomor polisi yang akan dihapus: ";
    cin >> cari;
    address del = findElm(L, cari);
    if (del == NULL) {
        cout << "Data tidak ditemukan!" << endl;
    } else {
        if (del == L.first)
            deleteFirst(L, del);
        else if (del == L.last)
            deleteLast(L, del);
        else {
            address prec = del->prev;
            deleteAfter(prec, del);
        }
        dealokasi(del);
        cout << "Data berhasil dihapus!\n";
    }

    cout << "\nData setelah penghapusan:\n";
    printInfo(L);

    return 0;
}

```

Screenshots Output

```

PS C:\Users\ASUS\Videos\strukdat\Laprak modul 6> .\main
Masukkan jumlah kendaraan: 3

Masukkan nomor polisi: D006
Masukkan warna kendaraan: Merah
Masukkan tahun kendaraan: 87

Masukkan nomor polisi: D003
Masukkan warna kendaraan: Hitam
Masukkan tahun kendaraan: 45

Masukkan nomor polisi: D001
Masukkan warna kendaraan: Ungu
Masukkan tahun kendaraan: 56

DATA LIST :
No Polisi : D006
Warna      : Merah
Tahun      : 87

No Polisi : D003
Warna      : Hitam
Tahun      : 45

No Polisi : D001
Warna      : Ungu
Tahun      : 56

```

Menampilkan Output berupa data mobil yang ingin diinputkan

```

Masukkan nomor polisi yang dicari: D003
Ditemukan!
No Polisi : D003
Warna      : Hitam
Tahun      : 45

```

Menampilkan output nomor polisi yang dicari

```

Masukkan nomor polisi yang akan dihapus: D001
Data berhasil dihapus!

Data setelah penghapusan:
DATA LIST :
No Polisi : D006
Warna      : Merah
Tahun      : 87

No Polisi : D003
Warna      : Hitam
Tahun      : 45

```

Menampilkan output setelah data dihapus

Deskripsi:

Kode program di atas merupakan implementasi struktur data doubly linked list dalam bahasa C++ yang digunakan untuk mengelola data kendaraan. Setiap elemen list menyimpan informasi berupa nomor polisi, warna, dan tahun pembuatan kendaraan, dengan node yang saling terhubung dua arah melalui pointer next dan prev. Program menyediakan berbagai operasi dasar seperti membuat list baru (createList), menambah elemen di akhir list (insertLast), mencari data berdasarkan nomor polisi (findElem), menampilkan seluruh isi list (printInfo), serta menghapus elemen di awal, akhir, atau di tengah list (deleteFirst, deleteLast, deleteAfter). Pada fungsi main, pengguna diminta untuk memasukkan sejumlah data kendaraan, kemudian program akan menampilkan daftar kendaraan yang tersimpan, melakukan pencarian berdasarkan nomor polisi, dan menghapus data kendaraan tertentu sesuai input pengguna, sebelum akhirnya menampilkan hasil akhir setelah penghapusan.

D. Kesimpulan

Kedua potongan kode di atas sama-sama menerapkan struktur data linked list ganda (doubly linked list), yaitu struktur data yang setiap elemennya (node) memiliki dua penunjuk — satu ke elemen sebelumnya (prev) dan satu ke elemen berikutnya (next). Perbedaannya, program pertama mengimplementasikan doubly linked list berbasis tipe data kompleks berupa struct kendaraan yang menyimpan atribut seperti nomor polisi, warna, dan tahun pembuatan. Program ini juga menyediakan operasi lengkap seperti membuat list, menambah data di akhir, mencari data berdasarkan nomor polisi, menampilkan seluruh isi list, serta menghapus node tertentu. Implementasi ini cocok untuk kasus nyata seperti pengelolaan data kendaraan dalam sistem administrasi parkir atau registrasi kendaraan.

Sementara itu, program kedua adalah versi yang lebih sederhana dan generik, hanya berfokus pada data bertipe int. Ia juga mengimplementasikan operasi dasar pada doubly linked list, seperti menambah node di awal, di akhir, atau di posisi tertentu; menghapus node pertama, terakhir, atau berdasarkan nilai tertentu; serta menampilkan isi list. Tujuan dari kode ini lebih ke arah demonstrasi konsep dasar manipulasi node dalam linked list ganda, bukan untuk aplikasi dunia nyata yang kompleks. Jadi, secara keseluruhan, perbedaan utama terletak pada tingkat kompleksitas data dan konteks penggunaannya — kode pertama bersifat aplikatif dan berorientasi objek data nyata, sedangkan kode kedua bersifat konseptual dan digunakan untuk memahami mekanisme dasar struktur data linked list.

E. Referensi

Linked list. Diakses dari: https://en.wikipedia.org/wiki/Linked_list

Introduction to Linked List. Diakses dari: <https://www.geeksforgeeks.org/introduction-to-linked-list-data-structure-and-algorithm-tutorials/>

Knuth, D. E. (1998). *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley.

Patel, P., & Dave, M. (2016). *A comparative study of linked list and array data structures*. *International Journal of Computer Applications*, 139(7), 1–5.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). *Data Structures and Algorithms in C++*. Wiley