LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLY LINKED LIST

Disusun Oleh :

NAMA : Taufik Hafit Zakaria

NIM : 103112430093

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Doubly Linked List merupakan salah satu bentuk struktur data dinamis yang merupakan pengembangan dari Single Linked List. Strukturnya tersusun atas kumpulan node (simpul) yang saling terhubung secara sekuensial dalam dua arah. Setiap node terdiri dari tiga bagian utama, yaitu data yang menyimpan informasi, pointer (next) yang menunjuk ke node berikutnya, dan pointer (prev) yang menunjuk ke node sebelumnya. Berbeda dengan array yang bersifat statis, Doubly Linked List bersifat fleksibel karena ukurannya dapat berubah secara dinamis selama program berjalan tanpa perlu mendeklarasikan ukuran awal.

Karakteristik Doubly Linked List

- Sifat Dinamis: Alokasi memori dilakukan saat program berjalan (runtime), sehingga mudah untuk menambah atau menghapus elemen tanpa perlu menggeser data lainnya.
- Dua Arah: Pointer pada setiap node menunjuk ke node berikutnya (next) dan node sebelumnya (prev). Hal ini memungkinkan penelusuran data dapat dilakukan dari node pertama (head) menuju akhir (tail), dan juga sebaliknya.
- Node Awal dan Akhir: Pointer prev pada node pertama selalu bernilai NULL, dan pointer next pada node terakhir selalu bernilai NULL. Keduanya menandakan batas awal dan akhir dari list.
- Operasi Dasar: Meliputi pembuatan list (create list), penyisipan elemen (insert), penghapusan elemen (delete), pencarian data (searching), dan pembaruan data (update). Implementasinya sedikit lebih kompleks karena harus mengelola dua pointer.

Keuntungan Doubly Linked List

- Traversal Dua Arah: Dapat menelusuri maju ke elemen berikutnya dan mundur ke elemen sebelumnya dengan mudah. Ini adalah keunggulan utamanya.
- Penghapusan Lebih Efisien: Proses menghapus sebuah node menjadi lebih mudah karena node sebelumnya dapat diakses secara langsung melalui pointer prev tanpa perlu iterasi dari awal.
- Fleksibel: Sama seperti Single Linked List, dapat menambah atau menghapus elemen tanpa perlu menggeser elemen lain seperti pada array.

Keterbatasan Doubly Linked List

- Overhead Memori Lebih Besar: Setiap node membutuhkan ruang tambahan untuk menyimpan pointer prev, sehingga penggunaan memorinya lebih besar dibandingkan Single Linked List.
- Akses Tidak Langsung: Untuk mengakses elemen tertentu, pencarian harus dimulai dari node pertama (head) atau terakhir (tail) karena tidak mendukung akses acak.
- Implementasi Lebih Kompleks: Operasi penyisipan dan penghapusan melibatkan lebih banyak langkah karena harus memperbarui lebih banyak pointer (next dan prev).

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

Guided 1

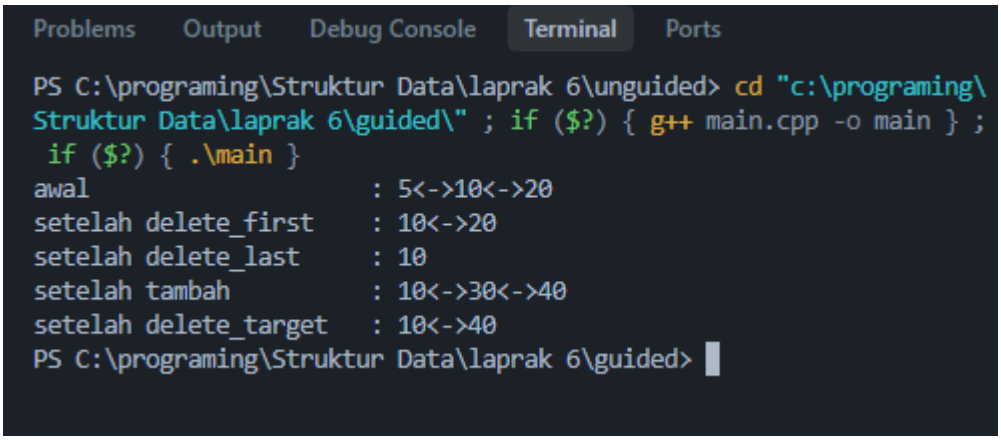
```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
#include <iostream>
using namespace std;

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;
}

```

```
35  void add_target(int targetValue, int newValue){
36      Node *current = ptr_first;
37      while(current != NULL && current->data != targetValue){
38          current = current->next;
39      }
40      if (current != NULL){
41          if (current == ptr_first){
42              add_last(newValue);
43          }else{
44              Node *newNode = new Node{newValue, current->prev, current};
45              current->prev->next = newNode;
46              current->prev = newNode;
47          }
48      }
49  }
50
51  void view(){
52      Node *current = ptr_first;
53      if (current == NULL){
54          cout << "List kosong!\n";
55          return;
56      }
57      while (current != NULL){
58          cout << current->data << (current->next != NULL ? "<->" : "");
59          current = current->next;
60      }
61      cout << endl;
62  }
63
```

```
64  void delete_first(){
65      if (ptr_first == NULL)
66          return;
67
68      Node *temp = ptr_first;
69
70      if (ptr_first == ptr_last){
71          ptr_first = NULL;
72          ptr_last = NULL;
73      }
74      else{
75          ptr_first = ptr_first->next;
76          ptr_first->prev = NULL;
77      }
78      delete temp;
79  }
80
81  void delete_last(){
82      if (ptr_last == NULL)
83          return;
84
85      Node *temp = ptr_last;
86
87      if (ptr_first == ptr_last){
88          ptr_first = NULL;
89          ptr_last = NULL;
90      }
91      else{
92          ptr_last = ptr_last->prev;
93          ptr_last->next = NULL;
94      }
95      delete temp;
96  }
97
```

```
98  void delete_target(int targetValue){
99      Node *current = ptr_first;
100     while (current != NULL && current->data != targetValue){
101         current = current->next;
102     }
103     if (current != NULL){
104         if (current == ptr_first){
105             delete_first();
106             return;
107         }else if (current == ptr_last){
108             delete_last();
109             return;
110         }else{
111             current->prev->next = current->next;
112             current->next->prev = current->prev;
113             delete current;
114         }
115     }
116 }
117
118 void edit_mode(int targetValue, int newValue){
119     Node *current = ptr_first;
120     while (current != NULL && current->data != targetValue){
121         current = current->next;
122     }
123     if (current != NULL){
124         current->data = newValue;
125     }
126 }
127
128 int main(){
129     add_first(10);
130     add_first(5);
131     add_last(20);
132     cout << "awal\t\t\t: ";
133     view();
134     delete_first();
135     cout << "setelah delete_first\t: ";
136     view();
137     delete_last();
138     cout << "setelah delete_last\t: ";
139     view();
140     add_last(30);
141     add_last(40);
142     cout << "setelah tambah\t\t: ";
143     view();
144     delete_target(30);
145     cout << "setelah delete_target\t: ";
146     view();
147     return 0;
148 }
```

Screenshots Output



Deskripsi:

Program yang saya buat ini adalah implementasi dari struktur data *Doubly Linked List* menggunakan C++. Intinya, setiap simpul (*node*) di dalam *list* ini tidak hanya punya penunjuk next untuk maju, tapi juga punya penunjuk prev untuk mundur. Kelebihan utamanya adalah kita bisa menelusuri data dari dua arah. Semua kode, dari struktur data hingga fungsi main, saya satukan dalam satu file agar lebih simpel. Saya juga sudah membuat fungsi-fungsi inti untuk mengelola *list* ini, seperti add_first, add_last untuk menambah data, delete_first, delete_last, dan delete_target untuk menghapus, serta view untuk menampilkan hasilnya.

Di dalam fungsi main, saya langsung menguji coba fungsi-fungsi tersebut secara berurutan. Awalnya, saya membuat *list* berisi 5 <-> 10 <-> 20. Lalu, saya coba hapus elemen depan (delete_first) dan elemen belakang (delete_last), sehingga yang tersisa hanya 10. Setelah itu, saya tambahkan lagi 30 dan 40 di akhir, membuat *list* menjadi 10 <-> 30 <-> 40. Terakhir, saya uji fungsi delete_target untuk menghapus nilai 30 yang ada di tengah. Hasil akhir yang ditampilkan program adalah 10 <-> 40.

Secara singkat, program ini menunjukkan bagaimana *pointer* prev dan next bekerja sama untuk menjaga agar *list* tetap terhubung dengan benar setiap kali ada perubahan data. Ini membuktikan keunggulan utama *Doubly Linked List*, yaitu kemudahan dalam operasi penghapusan di posisi manapun karena kita bisa langsung mengakses elemen sebelumnya.

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

Unguided 1

```
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
//Doublylist.h
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

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

#define Nil NULL

// ==== Deklarasi tipe data ====
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;
};

```

```
34  // ==== Deklarasi fungsi/prosedur ====
35  void createList(List &L);
36  address alokasi(infotype x);
37  void dealokasi(address &P);
38  void printInfo(List L);
39  void insertLast(List &L, address P);
40  address findElm(List L, string nopol);
41  void deleteFirst(List &L, address &P);
42  void deleteLast(List &L, address &P);
43  void deleteAfter(address Prec, address &P);
44
45  #endif
```

```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
//Doublylist.cpp
#include "Doublylist.h"

// Membuat list kosong
void createList(List &L) {
    L.first = Nil;
    L.last = Nil;
}

// Alokasi elemen baru
address alokasi(infotype x) {
    address P = new elmlist;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

// Dealokasi elemen
void dealokasi(address &P) {
    delete P;
    P = Nil;
}

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

```

```
// Insert di bagian akhir list
void insertLast(List &L, address P) {
    if (L.first == Nil) {
        L.first = P;
        L.last = P;
    } else {
        L.last->next = P;
        P->prev = L.last;
        L.last = P;
    }
}

// Mencari elemen berdasarkan nomor polisi
address findElm(List L, string nopol) {
    address P = L.first;
    while (P != Nil) {
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

// Menghapus elemen pertama
void deleteFirst(List &L, address &P) {
    if (L.first != Nil) {
        P = L.first;
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.first = L.first->next;
            L.first->prev = Nil;
            P->next = Nil;
        }
    }
}

```

```
78  // Menghapus elemen terakhir
79  void deleteLast(List &L, address &P) {
80      if (L.first != Nil) {
81          P = L.last;
82          if (L.first == L.last) {
83              L.first = Nil;
84              L.last = Nil;
85          } else {
86              L.last = L.last->prev;
87              L.last->next = Nil;
88              P->prev = Nil;
89          }
90      }
91  }
92
93  // Menghapus elemen setelah elemen tertentu
94  void deleteAfter(address Prec, address &P) {
95      if (Prec != Nil && Prec->next != Nil) {
96          P = Prec->next;
97          Prec->next = P->next;
98          if (P->next != Nil) {
99              P->next->prev = Prec;
100         }
101         P->next = Nil;
102         P->prev = Nil;
103     }
104 }
```

```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
//main-unguided.cpp
#include "Doublylist.h"
#include "Doublylist.cpp"

void tampilkanMenu() {
    cout << "1. Tambah Data" << endl;
    cout << "2. Cari Data" << endl;
    cout << "3. Hapus Data" << endl;
    cout << "4. Keluar Program" << endl;
    cout << "Pilih menu (0-1): ";
}

void tambahData(List &L) {
    infotype x;
    address P;
    string jawaban;

    cout << "\n=== TAMBAH DATA KENDARAAN ===" << endl;

    do {
        cout << endl;
        cout << "Masukkan nomor polisi: ";
        getline(cin, x.nopol);

        cout << "Masukkan warna kendaraan: ";
        getline(cin, x.warna);

        cout << "Masukkan tahun kendaraan: ";
        cin >> x.thnBuat;
        cin.ignore();

        if (findElm(L, x.nopol) != Nil) {
            cout << "\nError: Nomor polisi sudah terdaftar!" << endl;
            cout << endl;
            continue;
        }

        P = alokasi(x);
        insertLast(L, P);
        cout << endl;

        cout << "Apakah ingin menambahkan data kendaraan lagi? (ya/tidak): ";
        getline(cin, jawaban);

    } while (jawaban == "ya" || jawaban == "Ya" || jawaban == "YA" || jawaban == "Y" || jawaban == "y");

    cout << "\n=== DATA KENDARAAN ===" << endl;
    if (L.first == Nil) {
        cout << "Tidak ada data kendaraan." << endl;
    } else {
        printInfo(L);
    }
    cout << endl;
}

void cariData(List L) {
    if (L.first == Nil) {
        cout << "\nTidak ada data kendaraan untuk dicari." << endl;
        cout << endl;
        return;
    }

    string nopolCari;
    string lanjut;

    cout << "\n=== CARI DATA KENDARAAN ===" << endl;

```

```
    do {
        cout << endl;
        cout << "Masukkan nomor polisi yang dicari: ";
        getline(cin, nopolCari);

        address hasil = findElm(L, nopolCari);
        if (hasil != Nil) {
            cout << "\nData kendaraan ditemukan:" << endl;
            cout << "No polisi  : " << hasil->info.nopol << endl;
            cout << "Warna  : " << hasil->info.warna << endl;
            cout << "Tahun  : " << hasil->info.thnBuat << endl;
        } else {
            cout << "\nData tidak tersedia." << endl;
        }

        cout << endl;
        cout << "Apakah ingin melanjutkan pencarian? (ya/tidak): ";
        getline(cin, lanjut);

    } while (lanjut == "ya" || lanjut == "Ya" || lanjut == "YA" || lanjut == "Y" || lanjut == "y");

    cout << endl;
}

void hapusData(List &L) {
    if (L.first == Nil) {
        cout << "\nTidak ada data kendaraan untuk dihapus." << endl;
        cout << endl;
        return;
    }

    string nopolHapus;
    string lanjut;

    cout << "\n=== HAPUS DATA KENDARAAN ===" << endl;

    do {
        if (L.first == Nil) {
            cout << "\nTidak ada data kendaraan tersisa." << endl;
            break;
        }

        cout << endl;
        cout << "Masukkan nomor polisi yang ingin dihapus: ";
        getline(cin, nopolHapus);

        address hapus = findElm(L, nopolHapus);
        if (hapus != Nil) {
            address temp;

            if (hapus == L.first) {
                deleteFirst(L, temp);
            } else if (hapus == L.last) {
                deleteLast(L, temp);
            } else {
                address prec = hapus->prev;
                deleteAfter(prec, temp);
            }

```

```
            cout << "\nData dengan nomor polisi " << nopolHapus << " berhasil dihapus." << endl;
            dealokasi(temp);

            cout << endl;
            cout << "=== DATA KENDARAAN TERUPDATE ===" << endl;
            if (L.first == Nil) {
                cout << "Tidak ada data kendaraan." << endl;
            } else {
                printInfo(L);
            }
        } else {
            cout << "\nData dengan nomor polisi " << nopolHapus << " tidak ditemukan." << endl;
        }

        cout << endl;
        cout << "Apakah ingin melanjutkan penghapusan? (ya/tidak): ";
        getline(cin, lanjut);

    } while (lanjut == "ya" || lanjut == "Ya" || lanjut == "YA" || lanjut == "Y" || lanjut == "y");

    cout << endl;
}

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

    int pilihan;

    cout << "=== PROGRAM DATA KENDARAAN ===" << endl;
    cout << endl;

    do {
        tampilkanMenu();
        cin >> pilihan;
        cin.ignore();

        switch(pilihan) {
            case 1:
                tambahData(L);
                break;
            case 2:
                cariData(L);
                break;
            case 3:
                hapusData(L);
                break;
            case 4:
                cout << "\nTerima kasih telah menggunakan program ini!" << endl;
                break;
            default:
                cout << "\nPilihan tidak valid! Silakan pilih menu 0-1." << endl;
                cout << endl;
        }

    } while (pilihan != 4);

    // Dealokasi semua elemen
    address temp;
    while (L.first != Nil) {
        deleteFirst(L, temp);
        dealokasi(temp);
    }

    return 0;
}
```

Screenshots Output

```
=== TAMBAH DATA KENDARAAN ===

Masukkan nomor polisi: D001
Masukkan warna kendaraan: hitam
Masukkan tahun kendaraan: 90

Apakah ingin menambahkan data kendaraan lagi? (ya/tidak): y

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

Apakah ingin menambahkan data kendaraan lagi? (ya/tidak): y

Masukkan nomor polisi: D001
Masukkan warna kendaraan: merah
Masukkan tahun kendaraan: 80

Error: Nomor polisi sudah terdaftar!


Masukkan nomor polisi: D004
Masukkan warna kendaraan: kuning
Masukkan tahun kendaraan: 90

Apakah ingin menambahkan data kendaraan lagi? (ya/tidak): t

=== DATA KENDARAAN ===
No Polisi  : D001
Warna      : hitam
Tahun      : 90
------------------------
No Polisi  : D003
Warna      : putih
Tahun      : 70
------------------------
No Polisi  : D004
Warna      : kuning
Tahun      : 90
------------------------
```

```
=== CARI DATA KENDARAAN ===

Masukkan nomor polisi yang dicari: D001

Data kendaraan ditemukan:
No polisi  : D001
Warna      : hitam
Tahun      : 90

Apakah ingin melanjutkan pencarian? (ya/tidak): Y

Masukkan nomor polisi yang dicari: D006

Data tidak tersedia.
```

```
=== HAPUS DATA KENDARAAN ===

Masukkan nomor polisi yang ingin dihapus: D003

Data dengan nomor polisi D003 berhasil dihapus.

=== DATA KENDARAAN TERUPDATE ===
No Polisi  : D001
Warna      : hitam
Tahun      : 90
-----------------------
No Polisi  : D004
Warna      : kuning
Tahun      : 90
-----------------------

Apakah ingin melanjutkan penghapusan? (ya/tidak): Y

Masukkan nomor polisi yang ingin dihapus: D009

Data dengan nomor polisi D009 tidak ditemukan.
```

Deskripsi:

Program yang saya buat ini adalah implementasi dari struktur data Doubly Linked List menggunakan C++ untuk mengelola data kendaraan. Intinya, setiap simpul (node) di dalam list ini tidak hanya punya penunjuk next untuk maju, tapi juga punya penunjuk prev untuk mundur, jadi penelusuran data bisa dari dua arah. Struktur programnya saya pecah menjadi tiga file: Doublylist.h untuk deklarasi struktur dan fungsi, Doublylist.cpp untuk implementasi detail dari fungsi-fungsinya, dan main.cpp sebagai program utama yang berinteraksi dengan pengguna. Fungsi-fungsi inti yang saya buat meliputi createList, alokasi, insertLast untuk menambah data, findElm untuk pencarian, serta berbagai fungsi hapus seperti deleteFirst, deleteLast, dan deleteAfter.

Di dalam fungsi main, program ini berjalan secara interaktif dengan menampilkan sebuah menu kepada pengguna. Pengguna bisa memilih beberapa opsi, seperti: 1) Tambah Data Kendaraan, di mana program akan meminta input nomor polisi, warna, dan tahun, sambil memastikan tidak ada nomor polisi yang sama diinput dua kali. 2) Cari Data Kendaraan berdasarkan nomor polisi. 3) Hapus Data Kendaraan berdasarkan nomor polisi. Untuk proses penghapusan, program ini cukup pintar; setelah menemukan data, ia akan otomatis menentukan apakah harus menggunakan deleteFirst, deleteLast, atau deleteAfter tergantung posisi datanya. Program akan terus berjalan dalam sebuah loop sampai pengguna memilih opsi untuk keluar.

Secara keseluruhan, program ini adalah aplikasi nyata dari konsep Doubly Linked List untuk manajemen data. Ini menunjukkan bagaimana pointer prev dan next sangat berguna tidak hanya untuk navigasi dua arah, tapi juga untuk membuat operasi seperti penghapusan data di tengah list menjadi jauh lebih efisien. Dengan memisahkan deklarasi, implementasi, dan program utama ke dalam file yang berbeda, kode yang saya buat menjadi lebih rapi, terstruktur, dan mudah dikelola.

D. Kesimpulan

Berdasarkan praktikum yang telah dilaksanakan, dapat disimpulkan bahwa implementasi Doubly Linked List memberikan pemahaman mendalam mengenai manajemen memori dinamis dan manipulasi pointer. Proses pembuatan program pengelolaan data kendaraan telah berhasil menunjukkan penerapan konsep-konsep kunci, seperti alokasi dan dealokasi memori, serta penyambungan simpul (node) dari dua arah.

Kesimpulan dari praktikum ini adalah:

- Struktur Dua Arah: Doubly Linked List memungkinkan penelusuran maju dan mundur, yang sangat berguna dalam kasus di mana akses ke data sebelumnya diperlukan.

- Efisiensi Operasi: Operasi penghapusan, terutama untuk elemen yang tidak berada di awal list, menjadi jauh lebih efisien berkat akses langsung ke elemen sebelumnya melalui pointer prev.
- Kompleksitas: Dibandingkan Singly Linked List, Doubly Linked List memerlukan penanganan pointer yang lebih hati-hati pada setiap operasi penambahan dan penghapusan untuk menjaga integritas data.

E. Referensi

Drozdek, A. (2013). *Data Structures and Algorithms in C++* (4th ed.). Cengage Learning.

Mbejo, M. T., Nopa, L. A., Putri, J. S., & Risky, M. (2025). *Analisis struktur data linked list dalam pengolahan data mahasiswa*. Diakses pada tahun 2025 dari Jurnal Sains Informatika Terapan (JSIT), Vol. 2, No. 2.

Programiz. (2025). *C++ Program to Implement Doubly Linked List*. Diakses pada tahun 2025 dari https://www.programiz.com/dsa/doubly-linked-list