LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VI
DOUBLY LINKED LIST

Disusun Oleh :

NAMA : Damanik, Yohanes Geovan Ondova

NIM : 103112400022

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Doubly Linked list adalah *linked list* yang masing – masing elemen nya memiliki 2 *successar*, yaitu *successor* yang menunjuk sebelumnya (*prev*) dan *successor* pada elemen sesudahnya (*next*). Operasi pada *doubly linked list*:

- *Insert*:
  - *Insert First*: Menambahkan elemen baru di awal list.
  - *Insert Last*: Menambahkan elemen baru di akhir list.
  - *Insert After*: Menambahkan elemen baru setelah elemen tertentu.
  - *Insert Before*: Kebalikan dari *Insert after*, perbedannya terletak pada pencarian elemen
- *Delete*:
  - *Delete First*: Menghapus elemen pertama di list.
  - *Delete Last*: Menghapus elemen terakhir di list.
  - *Delete After*: Menghapus elemen setelah elemen tertentu.
  - *Delete Before*: Kebalikan dari *delete after*, perbedannya terltetak pada pencarian elemen.
- *Update, View, dan Searching*: Prosesnya mirip dengan *Singly linked list*, namun lebih mudah karena double linked list memungkinkan iterasi maju dan mundur.

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

Guided 1

Guided.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_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_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;

}
```

Output

```
PS C:\Users\Lenovo\Documents\STRUKTUR DATA MODUL 5> cd "c:\User
) { g++ Guided.cpp -o Guided } ; if ($?) { .\Guided }
Awal                    : 5 <-> 10 <-> 20
Setelah delete_first    : 5 <-> 10
Setelah delete_last     : 5
Setelah tambah          : 5 <-> 30 <-> 40
Setelah delete_target   : 5 <-> 40
PS C:\Users\Lenovo\Documents\STRUKTUR DATA MODUL 5>
```

Deskripsi:

Program ini merupakan implementasi struktur data doubly linked list (DLL) yang memungkinkan penyisipan node di awal (add_first), akhir (add_last), dan setelah node tertentu (add_target), serta penghapusan node di awal (delete_first), akhir (delete_last), dan berdasarkan nilai (delete_target), dilengkapi dengan fungsi menampilkan (view) dan mengubah data node (edit_node). Program demonstrasi dalam main menunjukkan operasi-operasi dasar dengan menambah node berisi nilai 5, 10, 20, kemudian menghapus node pertama dan terakhir, menambah nilai 30 dan 40, serta menghapus node berisi nilai 30, dimana seluruh proses dapat diamati melalui output berformat daftar terhubung dua arah.

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

Unguided 1

Doublylist.cpp

```
#include "Doublylist.h"

void CreateList(List &L) {
    L.first = Nil;
    L.last = Nil;
}

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

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

void printInfo(List L) {
    address P = L.first;
    if (L.first == Nil) {
        std::cout << "List kosong!" << std::endl;
    } else {
        while (P != Nil) {
            std::cout << "Nomor Polisi: " << P->info.nopol << std::endl;
            std::cout << "Warna       : " << P->info.warna << std::endl;
            std::cout << "Tahun       : " << P->info.thnBuat << std::endl;
            std::cout << std::endl;
            P = P->next;
        }
    }
}

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

```
address findElm(List L, std::string nopol) {
   address P = L.first;
   while (P != Nil) {
      if (P->info.nopol == nopol) {
         return P;
      }
      P = P->next;
   }
   return Nil;
}

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

void deleteLast(List &L, address &P) {
   P = L.last;
   if (L.first == L.last) {
      L.first = Nil;
      L.last = Nil;
   } else {
      L.last = P->prev;
      L.last->next = Nil;
      P->prev = Nil;
   }
   dealokasi(P);
}

void deleteAfter(List &L, address &Prec, address &P) {

   P = Prec->next;
   if (P == L.last) {
      deleteLast(L, P);
   } else {
      Prec->next = P->next;
      P->next->prev = Prec;
      P->next = Nil;
      P->prev = Nil;
      dealokasi(P);
   }
```

```
}
```

Doublyist.h

```
#ifndef Doublylist_H
#define Doublylist_H

#include <iostream>
#include <string>

#define Nil NULL

struct infotype {
   std::string nopol;
   std::string warna;
   int thnBuat;
};

typedef struct elmlist *address;

struct elmlist {
   infotype info;
   address next;
   address prev;
};

struct List {
   address first;
   address last;
};

/** Deklarasi Prosedur dan Fungsi **/

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, std::string nopol);

void deleteFirst(List &L, address &P);

void deleteLast(List &L, address &P);
```

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

#endif
```

Main.cpp

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

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

    infotype dataKendaraan;
    address P;



    dataKendaraan = {"D001", "hitam", 90};
    P = alokasi(dataKendaraan);
    insertLast(L, P);


    dataKendaraan = {"D003", "putih", 70};
    P = alokasi(dataKendaraan);
    insertLast(L, P);


    std::cout << "masukkan nomor polisi: D001" << std::endl;
    if (findElm(L, "D001") != Nil) {
        std::cout << "nomor polisi sudah terdaftar" << std::endl << std::endl;
    }


    dataKendaraan = {"D004", "kuning", 90};
    P = alokasi(dataKendaraan);
    insertLast(L, P);


    std::cout << "\nDATA LIST 1" << std::endl;
    printInfo(L);
    std::cout << "-----------------------------------" << std::endl;


    std::string cariNopol = "D001";
    std::cout << "\nMasukkan Nomor Polisi yang dicari: " << cariNopol << std::endl;
    address ditemukan = findElm(L, cariNopol);
```

```
    if (ditemukan != Nil) {
        std::cout << "Data ditemukan:" << std::endl;
        std::cout << "Nomor Polisi: " << ditemukan->info.nopol << std::endl;
        std::cout << "Warna      : " << ditemukan->info.warna << std::endl;
        std::cout << "Tahun      : " << ditemukan->info.thnBuat << std::endl;
    } else {
        std::cout << "Data dengan nomor polisi " << cariNopol << " tidak ditemukan."
<< std::endl;
    }
    std::cout << "-----------------------------------" << std::endl;


    std::string nopolHapus = "D003";
    std::cout << "\nMasukkan Nomor Polisi yang akan dihapus : " << nopolHapus <<
std::endl;

    address elmHapus = findElm(L, nopolHapus);
    if (elmHapus != Nil) {
        address Pdel;
        if (elmHapus == L.first) {
            deleteFirst(L, Pdel);
        } else if (elmHapus == L.last) {
            deleteLast(L, Pdel);
        } else {
            address Prec = elmHapus->prev;
            deleteAfter(L, Prec, Pdel);
        }
        std::cout << "Data dengan nomor polisi " << nopolHapus << " berhasil
dihapus." << std::endl;
    } else {
        std::cout << "Data dengan nomor polisi " << nopolHapus << " tidak
ditemukan." << std::endl;
    }
    std::cout << "-----------------------------------" << std::endl;


    std::cout << "\nDATA LIST 1" << std::endl;
    printInfo(L);
    std::cout << "-----------------------------------" << std::endl;

    return 0;
}
```

Output

```
PS C:\Users\Lenovo\Documents\STRUKTUR DATA MODUL 5> cd "c:\Users\Lenovo
) { g++ main.cpp -o main } ; if ($?) { .\main }
masukkan nomor polisi: D001
nomor polisi sudah terdaftar


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

Nomor Polisi: D003
Warna       : putih
Tahun       : 70

Nomor Polisi: D004
Warna       : kuning
Tahun       : 90

-----------------------------------

Masukkan Nomor Polisi yang dicari: D001
Data ditemukan:
Nomor Polisi: D001
Warna       : hitam
Tahun       : 90
-----------------------------------
```

```
Masukkan Nomor Polisi yang dicari: D001
Data ditemukan:
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

-----------------------------------
PS C:\Users\Lenovo\Documents\STRUKTUR DATA MODUL 5>
```

Deskripsi:

Program ini merupakan implementasi struktur data doubly linked list untuk mengelola data kendaraan yang terdiri dari nomor polisi, warna, dan tahun pembuatan, dimana program dapat membuat list, mengalokasikan node baru, menambahkan data di akhir list, mencari data berdasarkan nomor polisi, serta menghapus data di awal, akhir, atau setelah

node tertentu. Pada fungsi main, program mendemonstrasikan penggunaan dengan menambahkan tiga data kendaraan, melakukan pencarian dan pengecekan duplikat data "D001", menghapus data "D003" dengan penanganan kasus penghapusan di berbagai posisi, serta menampilkan seluruh data sebelum dan setelah penghapusan untuk menunjukkan integritas struktur data.

D. Kesimpulan

Praktikum ini dapat mengimplementasi struktur data untuk menunjukkan kemampuan dalam mengelola data secara dinamis dengan operasi penyisipan di akhir list, pencarian berdasarkan nomor polisi, serta penghapusan elemen di berbagai posisi (awal, akhir, dan tengah) dengan memanfaatkan pointer prev dan next yang memungkinkan traversal dua arah, sehingga membuktikan keunggulan doubly linked list dalam efisiensi operasi penghapusan dan fleksibilitas manipulasi data dibandingkan singly linked list.

E. Referensi

https://repository.unikom.ac.id/38651/1/Bab%20VI%20-%20Double%20Linked%20List.pdf

https://www.programiz.com/dsa/doubly-linked-list