LAPORAN PRAKTIKUM

MODUL 6

“DOUBLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

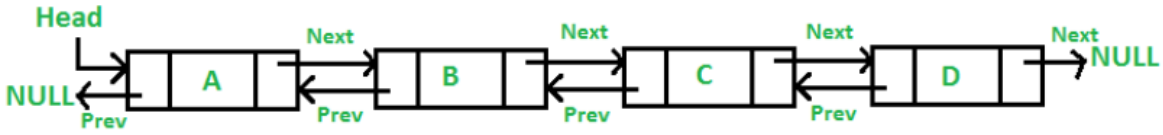
1. Tujuan

- Memahami konsep modul linked list.
- Mengapilkasikan konsep double linked list dengan menggunakan pointer dan dengan Bahasa C.

2. Landasan Teori

Double linked list adalah sebuah daftar dua arah di mana setiap simpul memiliki dua penunjuk, yaitu penunjuk berikutnya dan penunjuk sebelumnya, yang masing-masing mengacu ke simpul berikutnya dan simpul sebelumnya. Tidak seperti single linked list di mana setiap node hanya menunjuk ke node berikutnya, double linked list memiliki penunjuk ekstra sebelumnya yang memungkinkan penjelajahan dalam arah maju dan mundur.

Dalam sebuah double linked list, dua buah penunjuk (pointer) dipertahankan untuk melacak senarai tersebut, yaitu kepala (yang menunjuk pada simpul pertama) dan ekor (yang menunjuk pada simpul terakhir).



double linked list

Untuk membuat node dalam double link list di C++, setiap simpul dari sebuah daftar berantai ganda (DLL) terdiri dari tiga field:

- Item atau Data: Ini adalah nilai yang disimpan di dalam simpul.
- Next: Referensi ke simpul berikutnya dalam daftar.
- Prev: Referensi ke simpul terakhir dalam daftar.

Dalam C++, double linked list dapat dibuat dengan menggunakan format di bawah ini:

```
class Node
{
        public:
        int data;
        Node *next;
        Node *prev;
};
```

Dalam sebuah doube linked list, ada tiga cara untuk melakukan operasi insert:

- Insert first
  Menambahkan node di awal doubly linked list melibatkan pembuatan node baru dan penyesuaian pointer. Jika linked list masih kosong, node baru akan menjadi satu-satunya elemen. Namun, jika sudah ada elemen, node baru akan ditempatkan di posisi paling depan. Untuk itu, pointer next dari node baru akan mengarah ke node yang sebelumnya menjadi head,

sementara pointer prev dari node head sebelumnya akan diubah mengarah ke node baru. Dengan demikian, node baru secara resmi menjadi head dari linked list, menempati posisi paling awal dalam struktur data tersebut.

- Insert last
  Menambahkan node di akhir doubly linked list melibatkan pembuatan node baru dan penyesuaian pointer. Jika linked list masih kosong, node baru langsung menjadi head sekaligus tail. Namun, jika sudah ada elemen, kita akan mencari node terakhir (tail). Setelah ditemukan, pointer next dari node tail akan diarahkan ke node baru, sementara pointer prev dari node baru akan mengarah ke node tail sebelumnya. Dengan demikian, node baru menjadi node terakhir yang terhubung dalam linked list, memperpanjang struktur data tersebut.
- Insert after
  Menambahkan node setelah node tertentu dalam doubly linked list melibatkan beberapa langkah. Pertama, kita membuat node baru yang akan disisipkan. Kemudian, kita mencari node yang menjadi rujukan untuk penempatan node baru ini. Setelah ditemukan, kita menghubungkan node baru dengan node-node di sekitarnya. Jika node target adalah node terakhir, node baru akan menjadi node terakhir yang baru, sehingga pointer next-nya akan bernilai null. Namun, jika node target bukan node terakhir, maka node baru akan berada di antara node target dan node setelahnya. Oleh karena itu, pointer-pointer next dan prev dari node-node yang terlibat perlu disesuaikan agar membentuk hubungan yang benar dalam linked list. Dengan demikian, node baru berhasil disisipkan pada posisi yang diinginkan.
- Insert before
  Menambahkan node sebelum node tertentu dalam doubly linked list melibatkan penyesuaian pointer yang cermat. Pertama, kita siapkan node baru yang akan disisipkan. Kemudian, kita mencari node yang menjadi acuan untuk penempatan node baru ini. Jika node target adalah node pertama (head), maka node baru akan menjadi head yang baru, sehingga pointer prev-nya akan bernilai null. Namun, jika node target bukan head, maka node baru akan berada di antara node sebelumnya dan node target. Untuk itu, kita perlu mengatur pointer next dan prev dari node-node yang terlibat agar terhubung dengan benar. Dengan demikian, node baru berhasil disisipkan pada posisi yang diinginkan, mengubah urutan elemen dalam linked list.

Dalam double linked list terdapat juga operasi *delete* berfungsi untuk menghapus node pada posisi tertentu, seperti di awal, di akhir, setelah node tertentu, atau sebelum node tertentu. Karena setiap node memiliki dua pointer (ke node berikutnya dan node sebelumnya), kita harus memperbarui kedua pointer ini saat

menghapus node. Ada beberapa jenis operasi delete dalam double linked list seperti berikut:

- Delete First
  Menghapus node pertama pada doubly linked list melibatkan beberapa kondisi. Jika linked list kosong, tidak ada tindakan yang perlu dilakukan. Jika hanya ada satu node, maka node tersebut langsung dihapus dan head menjadi null. Namun, jika terdapat lebih dari satu node, kita akan menyimpan node head saat ini ke dalam variabel sementara. Kemudian, head akan diubah menjadi node berikutnya. Pointer prev dari node head yang baru ini akan diset menjadi null. Terakhir, node yang sebelumnya disimpan dalam variabel sementara (node yang telah diputus dari linked list) akan dihapus dari memori. Dengan demikian, node pertama berhasil dihapus dan struktur linked list tetap terjaga.
- Delete Last
  Menghapus node terakhir pada doubly linked list melibatkan beberapa langkah. Jika linked list kosong atau hanya berisi satu node, maka penghapusan dilakukan dengan cara yang sederhana. Namun, jika terdapat lebih dari satu node, kita perlu mencari node terakhir terlebih dahulu. Setelah ditemukan, node kedua terakhir akan diputuskan hubungannya dengan node terakhir dengan mengatur pointer next-nya menjadi null. Kemudian, node terakhir yang telah terlepas dari linked list akan dihapus dari memori. Dengan demikian, node terakhir berhasil dihapus dan struktur linked list tetap valid.
- Delete After
  Menghapus node setelah node tertentu dalam doubly linked list melibatkan beberapa langkah. Pertama, kita perlu memastikan bahwa node yang menjadi acuan (prevNode) valid dan bukan merupakan node terakhir. Jika kondisi ini terpenuhi, kita akan menyimpan node yang akan dihapus ke dalam variabel sementara. Kemudian, kita menghubungkan node sebelum node yang akan dihapus (prevNode) dengan node sesudahnya (prevNode->next->next). Jika node yang dihapus bukan node terakhir, maka pointer prev dari node yang sebelumnya mengikuti node yang dihapus akan diubah mengarah ke prevNode. Terakhir, node yang telah diisolasi dari linked list akan dihapus dari memori. Dengan demikian, node yang berada setelah node tertentu berhasil dihapus dan struktur linked list tetap terjaga.
- Delete Before
  Menghapus node sebelum node tertentu dalam doubly linked list melibatkan beberapa langkah. Pertama, kita perlu memastikan bahwa node yang menjadi acuan (nextNode) valid dan bukan merupakan node pertama. Jika kondisi ini terpenuhi, kita akan menyimpan node yang akan dihapus ke dalam variabel sementara. Kemudian, kita menghubungkan node setelah node yang akan dihapus (nextNode) dengan node sebelumnya (nextNode->prev->prev). Jika node yang dihapus bukan node

pertama, maka pointer next dari node yang sebelumnya mendahului node yang dihapus akan diubah mengarah ke nextNode. Terakhir, node yang telah diisolasi dari linked list akan dihapus dari memori. Dengan demikian, node yang berada sebelum node tertentu berhasil dihapus dan struktur linked list tetap terjaga.

Operasi update, view, dan searching sering kali digunakan untuk mengelola dan menemukan data dalam struktur data. Berikut penjelasan mengenai operasi tersebut;

- Update, merupakan pembaruan pada double linked list dilakukan untuk mengubah nilai data di dalam sebuah node tertentu**.** Jika linked list dalam keadaan kosong, tentu saja tidak ada data yang dapat diubah. Untuk memperbarui data, kita perlu melakukan traversal atau penelusuran dimulai dari node pertama (head) untuk menemukan node yang ingin diubah nilainya. Pencarian ini biasanya dilakukan dengan membandingkan data pada setiap node atau dengan mengetahui posisi node yang ingin diubah. Setelah node target ditemukan, maka nilai data di dalam node tersebut dapat diganti dengan nilai baru yang diinginkan. Dengan demikian, data dalam linked list dapat dijaga agar selalu relevan dan up-to-date.
- View, memungkinkan untuk melihat seluruh isi dari sebuah double linked list. Prosesnya dimulai dari node pertama (head) dan kemudian secara berurutan mengunjungi setiap node berikutnya. Pada setiap node yang dikunjungi, data yang tersimpan di dalamnya akan ditampilkan. Proses ini berlanjut hingga mencapai node terakhir, yaitu node yang pointer next-nya bernilai null. Dengan demikian, kita dapat memperoleh gambaran lengkap mengenai data-data apa saja yang tersimpan dalam linked list tersebut, mulai dari awal hingga akhir. Operasi ini sangat berguna untuk melakukan pengecekan, debugging, atau sekadar menampilkan informasi yang tersimpan dalam struktur data tersebut.
- Searching, operasi pencarian pada double linked list bertujuan untuk menemukan sebuah node yang mengandung nilai data tertentu. Karena struktur linked list yang tidak memungkinkan akses langsung ke elemen tertentu, maka pencarian dilakukan secara berurutan mulai dari node pertama (head). Setiap node akan diperiksa satu per satu dengan membandingkan nilai datanya dengan nilai yang sedang dicari. Jika nilai data yang dicari ditemukan, maka proses pencarian akan berhenti dan biasanya kita akan mendapatkan informasi mengenai posisi atau node yang berisi data tersebut. Namun, jika setelah menelusuri seluruh node tidak ditemukan kecocokan, maka dapat disimpulkan bahwa data yang dicari tidak ada dalam linked list.

3. Guided

Program ini merupakan implementasi Double Linked List yang mencakup operasi dasar seperti penambahan, penghapusan, pembaruan, dan penampilan data. Struktur data ini bermanfaat ketika kita perlu menyimpan dan mengelola data yang dapat diakses secara efisien dari kedua arah.

Pertama-tama dibuat kelas node untuk merepresentasikan elemen dari linked list. dengan atribut, data, prev, dan next. Setelahnya dibuat kelas DoublyLinkedList merepresentasikan seluruh linked list dan memiliki dua atribut utama yaitu, head dan tail.

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
```

Pada kelas DoublyLinkedList memiliki beberapa fungsi yang dijalankan sepert,

- Konstruktor yang menginisialisasi head dan tail menjadi nullptr, yang berarti list kosong saat pertama kali dibuat.
- Fungsi insert, untuk menambahkan elemen baru di depan list.
- Fungsi deleteNode untuk menghapus elemen dari depan list.
- Fungsi update yang digunakan mencari node yang memiliki data sama dengan oldData dan menggantinya dengan newData.
- Fungsi deleteAll untuk menghapus semua elemen dalam list.
- Fungsi display untuk menampilkan semua elemen dalam list.

```
// construktor untuk inisialisasi head dan tail
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    // fungsi menambahkan elemen di depan List
    void insert(int data){
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr){
            head->prev = newNode;
        } else {
            tail = newNode;
        }
        head = newNode;
    }

    // fungsi untuk menghapus elemen dari depan list
    void deleteNode(){
        if (head == nullptr){
            return;
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr){
        } else {
            tail = nullptr;
        }
        delete temp;
    }

    // Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menampilkan semua elemen di list
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};
```

Fungsi main berfungsi sebagai antarmuka pengguna untuk berinteraksi dengan linked list. Ini memberikan menu kepada pengguna untuk memilih tindakan yang diinginkan. Pengguna dapat memasukkan pilihan mereka dan program akan menjalankan fungsi yang sesuai berdasarkan pilihan tersebut. Ini merupakan contoh penggunaan struktur data yang interaktif.

```cpp
int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
                list.insert(data);
                break;
            }
            case 2: {
                list.deleteNode();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                bool updated = list.update(oldData, newData);
                if (!updated) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
}
```

Output dari program akan menjadi seperti berikut:

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 10
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
10
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
```

4. Unguided

1. Program ini merupakan implementasi dari struktur data double linked list. Program ini menyimpan informasi tentang kendaraan, termasuk nomor polisi, warna, dan tahun pembuatan. Dibuat 3 class yaitu doublelist.h, doublelist.cpp, dan main.cpp.

   Doublelist.h
   File ini adalah header file yang mendefinisikan struktur data dan fungsi-fungsi yang akan digunakan dalam program. Struktur kendaraan menyimpan informasi tentang kendaraan, termasuk nomor polisi (nopol), warna (warna), dan tahun pembuatan (thnBuat). Tipe Data infotype, Alias untuk tipe data kendaraan, memudahkan penggunaan tipe ini dalam kode. Struktur ElmList merepresentasikan elemen dari daftar ganda. Setiap elemen memiliki informasi kendaraan dan dua pointer (ke elemen berikutnya dan sebelumnya). Struktur List merepresentasikan daftar ganda itu sendiri, dengan pointer ke elemen pertama dan terakhir. Tipe Data address, alias untuk pointer ke ElmList, digunakan untuk mengelola elemen-elemen dalam daftar.

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H
#include <string>
using namespace std;

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;
};

void CreateList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(List L);
void insertLast(List &L, address P);

#endif
```

Doublelist.cpp

File ini berisi implementasi dari fungsi-fungsi yang dideklarasikan di dalam header file. Seperti fungsi-fungsi berikut:

-CreateList: Mengatur pointer First dan Last menjadi NULL

-alokasi: Mengalokasikan memori untuk elemen baru dari daftar, mengisi informasi kendaraan, dan mengatur pointer next dan prev ke NULL.

-dealokasi: Menghapus elemen dari memori dan mengatur pointer menjadi NULL untuk mencegah akses ke memori yang sudah dibebaskan.

-printInfo: Mencetak detail setiap kendaraan dalam daftar

-insertLast: Menambahkan elemen baru di akhir daftar.

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

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 << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.thnBuat << endl;
        if (P->next != NULL) {
            cout << endl;
        }
        P = P->next;
    }
}

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

Main.cpp

File ini adalah titik masuk program. Di sini, pengguna dapat memasukkan data kendaraan dan melihat hasilnya.

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

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

    infotype x;

    // Input first vehicle
    cout << "masukkan nomor polisi: ";
    x.nopol = "D001";
    cout << x.nopol << endl;
    cout << "masukkan warna kendaraan: ";
    x.warna = "hitam";
    cout << x.warna << endl;
    cout << "masukkan tahun kendaraan: ";
    x.thnBuat = 90;
    cout << x.thnBuat << endl;
    cout << endl;

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

    // Input second vehicle
    cout << "masukkan nomor polisi: ";
    x.nopol = "D003";
    cout << x.nopol << endl;
    cout << "masukkan warna kendaraan: ";
    x.warna = "putih";
    cout << x.warna << endl;
    cout << "masukkan tahun kendaraan: ";
    x.thnBuat = 70;
    cout << x.thnBuat << endl;
    cout << endl;

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

    // Input third vehicle
    cout << "masukkan nomor polisi: ";
    x.nopol = "D001";
    cout << x.nopol << endl;
    cout << "masukkan warna kendaraan: ";
    x.warna = "merah";
    cout << x.warna << endl;
    cout << "masukkan tahun kendaraan: ";
    x.thnBuat = 80;
    cout << x.thnBuat << endl;
    cout << "nomor polisi sudah terdaftar" << endl;
    cout << endl;

    // Input fourth vehicle
    cout << "masukkan nomor polisi: ";
    x.nopol = "D004";
    cout << x.nopol << endl;
    cout << "masukkan warna kendaraan: ";
    x.warna = "kuning";
    cout << x.warna << endl;
    cout << "masukkan tahun kendaraan: ";
    x.thnBuat = 90;
    cout << x.thnBuat << endl;
    cout << endl;

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

    printInfo(L);

    return 0;
}
```

Output dari program:

```
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 LIST 1
no polisi : D001
warna     : hitam
tahun     : 90

no polisi : D003
warna     : putih
tahun     : 70

no polisi : D004
warna     : kuning
tahun     : 90
```

2. Pada program ini masih menggunakan program yang sama hanya menambahkan pencarian nomor polisi.
   Doublelist.h
   Pada file header ditambahkan fungsi untuk mencari elemen dalam sebuah list yang digunakan untuk mencari dan mengembalikan alamat dari elemen yang dicari. Kemudian, dibuat sebuah prosedur untuk mencari dan mencetak data berdasarkan nomor polisi.

```
address findElm(List L, infotype x);  // 2
void searchAndPrint(List L, string searchNopol);  // 2
```

   Doublelist.cpp
   Dari file header tadi di deklarasikan fungsi yang telah dipanggil tadi pad afile cpp. Fungsi findELm mencari sebuah data dalam List dan mengembalikan alamat dimana data tersebut berada. Jika data tidak ditemukan, mengembalikan NULL. Kemudian prosedur searchAndPrint digunakan untuk mencetak hasil pencarian dan tidak mengembalikan nilai (void).

```
// New function to find element
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;
}

// New helper function to search and print results
void searchAndPrint(List L, string searchNopol) {
    cout << "Masukkan Nomor Polisi yang dicari : " << searchNopol << endl;

    infotype searchData;
    searchData.nopol = searchNopol;

    address found = findElm(L, searchData);

    if (found != NULL) {
        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;
    }
}
```

Main.cpp

Pada fungsi utama ditambahkan pemanggilan prosedur searchAndPrint, dimana fungsi ini akan Mencari kendaraan dengan nomor polisi "D001" dalam List L.

```
// 2
searchAndPrint(L, "D001");
```

Output dari program akan menjadi seperti ini:

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

3. Masih sama dengan program sebelumnya pada program ini ditambahkan beberapa prosedur untuk menghapus menggunakan prosedur delete first, delete last, dan delete after

   Doublelist.h

   Pada file header diimplementasikan prosedur untuk menghapus elemen pertama dari linked list, menghapus elemen terakhir dari linked list, menghapus elemen setelah node tertentu, dan menghapus nilai tertentu.

```
// 3
void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(address Prec, address &P);
void deleteElement(List &L, string nopol);
```

   Doublelist.cpp

   Pada file cpp diimplementasikan operasi penghapusan pada double linked list dengan mempertimbangkan berbagai posisi node yang akan dihapus sesuai dengan file header, prodedur deleteFirst menghapus node pertama dengan

menyimpan alamatnya di P, prosedur deleteLast Menghapus node terakhir dengan menyimpan alamatnya di P, prosedur deleteAfter Menghapus node setelah Prec, prosedur deleteElement Mencari node dengan nomor polisi yang diinput

```
// 3
void deleteFirst(List &L, address &P) {
    P = L.First;
    if (L.First->next == NULL) {
        L.First = NULL;
        L.Last = NULL;
    } else {
        L.First = L.First->next;
        L.First->prev = NULL;
        P->next = NULL;
    }
}

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

```
void deleteAfter(address Prec, address &P) {
    P = Prec->next;
    Prec->next = P->next;
    if (P->next != NULL) {
        P->next->prev = Prec;
    }
    P->next = NULL;
    P->prev = NULL;
}

void deleteElement(List &L, string nopol) {
    cout << "Masukkan Nomor Polisi yang akan dihapus : " << nopol << endl;

    address P;
    infotype x;
    x.nopol = nopol;
    address found = findElm(L, x);

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

Main.cpp

Pada fungsi utama memanggil fungsi deleteElement untuk menghapus data dengan nomor polisi "D003" dari List L, Jika ditemukan, node akan dihapus sesuai posisinya (awal/akhir/tengah). Lalu ditampilkan label “DATA LIST 1”, kemudian memanggil fungsi printInfo untuk menampilkan seluruh data yang ada pada List L.

```
// 3
deleteElement(L, "D003");

cout << endl << "DATA LIST 1" << endl;
printInfo(L);
```

Output yang dikeluarkan akan menjadi seperti ini:

```
Masukkan Nomor Polisi yang akan dihapus : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1
DATA LIST 1
no polisi : D001
warna     : hitam
tahun     : 90

no polisi : D004
warna     : kuning
tahun     : 90
```

5. Kesimpulan

Double Linked List, memungkinkan akses dan manipulasi data dari kedua arah. Praktikum ini mengimplementasikan struktur ini untuk mengelola data kendaraan, dengan fitur lengkap seperti penambahan, pencarian, penghapusan, dan tampilan data. Keunggulan utama Double Linked List adalah kemudahan dalam melakukan operasi pada awal dan akhir list.