LAPORAN PRAKTIKUM

MODUL

DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

a. Memahami konsep modul linked list.

b. Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C.

II. LANDASAN TEORI

1. *Double Linked List*

Double Linked List (DLL) adalah salah satu jenis struktur data berbasis daftar yang terdiri dari rangkaian elemen atau node yang terhubung secara berurutan. Setiap node dalam DLL menyimpan dua buah pointer: satu mengarah ke node sebelumnya (*prev*) dan satu lagi mengarah ke node berikutnya (*next*). Hal ini berbeda dari *single linked list*, di mana setiap node hanya memiliki satu pointer yang mengarah ke node berikutnya. Struktur ini memungkinkan traversal dua arah, baik maju (dari awal ke akhir) maupun mundur (dari akhir ke awal), yang meningkatkan fleksibilitas manipulasi data.

Komponen *Double Linked List*

1. *First*: Pointer ini menunjuk ke node pertama dalam list. Elemen pertama adalah pintu masuk utama dalam operasi traversal dari depan.
2. *Last*: Pointer ini menunjuk ke node terakhir dalam list. Elemen terakhir memungkinkan traversal dimulai dari akhir.
3. *Next*: Pointer yang digunakan untuk mengakses node berikutnya dalam urutan maju.
4. *Prev*: Pointer yang digunakan untuk mengakses node sebelumnya dalam urutan mundur.

Setiap node dalam DLL terdiri dari tiga bagian: informasi yang disimpan (data), pointer *next*, dan pointer *prev*. Dengan demikian, sebuah node memiliki akses ke kedua arah, baik ke depan maupun ke belakang, melalui dua pointer ini. Hal ini membuat DLL lebih efisien ketika diperlukan akses secara dua arah dibandingkan dengan single linked list yang hanya mendukung akses maju.

Operasi Dasar pada *Double Linked List*

Berikut adalah beberapa operasi dasar yang dapat dilakukan pada *Double Linked List*:

1. *Insert* (Menambahkan Elemen)
    - *Insert First*: Operasi ini menambahkan elemen baru di bagian awal list. Untuk melakukannya, elemen baru akan ditempatkan sebelum elemen pertama yang ada. Jika list masih kosong, elemen baru ini akan menjadi elemen pertama dan terakhir. Proses ini membutuhkan pengaturan ulang pointer *first* dan *prev* dari elemen pertama sebelumnya.
    - *Insert Last*: Operasi ini menambahkan elemen di akhir list. Elemen baru akan menjadi elemen terakhir, dan pointer last akan diperbarui untuk menunjuk ke elemen baru ini. Proses ini juga mengatur ulang pointer *next* dari elemen terakhir sebelumnya.

- *Insert After*: Operasi ini menyisipkan elemen baru setelah node tertentu dalam list. Proses ini melibatkan perubahan pointer *next* dari elemen yang disisipkan dan pointer *prev* dari elemen berikutnya.
- *Insert Before*: Operasi ini menyisipkan elemen baru sebelum node tertentu. Mirip dengan Insert After, operasi ini melibatkan pengaturan ulang pointer *prev* dari elemen yang disisipkan dan pointer *next* dari elemen sebelumnya.

2. *Delete* (Menghapus Elemen)
   - *Delete First*: Operasi ini menghapus elemen pertama dari list. Setelah elemen pertama dihapus, pointer *first* akan diarahkan ke elemen berikutnya, dan pointer *prev* dari elemen baru pertama akan diatur ke NULL.
   - *Delete Last*: Operasi ini menghapus elemen terakhir dari list. Setelah elemen terakhir dihapus, pointer *last* akan diarahkan ke elemen sebelumnya, dan pointer *next* dari elemen baru terakhir akan diatur ke NULL.
   - *Delete After*: Operasi ini menghapus elemen setelah elemen tertentu. Proses ini melibatkan perubahan pointer *next* dari elemen yang dihapus dan mengatur ulang pointer *prev* dari elemen berikutnya.
   - *Delete Before*: Operasi ini kebalikan dari *Delete After*, yaitu menghapus elemen sebelum elemen tertentu. Proses ini mengatur ulang pointer dari elemen sebelumnya serta elemen setelahnya.

III. GUIDED

1. guided

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

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

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

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

    // Fungsi untuk 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; // Jika list kosong, tail juga mengarah ke node baru
        }
        head = newNode;
    }

    // Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
```

```
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // 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;
    }
};
```

```
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;
```

```
        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;
            }
        }
    }
```

Program Program ini mengimplementasikan struktur data *Doubly Linked List* (DLL) menggunakan kelas-kelas dan metode-metode dalam bahasa C++. Pada dasarnya, *Doubly Linked List* adalah daftar yang terdiri dari node-node yang saling terhubung di mana setiap node memiliki dua penunjuk (pointer): satu ke elemen sebelumnya (*prev*) dan satu ke elemen berikutnya (*next*). Dengan adanya dua pointer ini, program dapat bergerak maju dan mundur dalam daftar.

- **Kelas Node**

  Setiap elemen atau node dalam *Doubly Linked List* terdiri dari dua bagian utama:

  1. Data: Ini adalah informasi yang disimpan dalam node, dalam hal ini berupa tipe integer.
  2. Pointer:
     - *prev*: Mengarah ke elemen sebelumnya.
     - *next*: Mengarah ke elemen berikutnya.

  Node ini akan menjadi bagian fundamental dari list, karena setiap data yang ditambahkan, dihapus, atau diupdate akan disimpan dalam objek Node.

- **Kelas DoublyLinkedList**

  Kelas ini berfungsi sebagai representasi utama dari *Doubly Linked List* dan memiliki dua pointer utama:

  1. *head*: Pointer ini selalu menunjuk ke elemen pertama dari list.
  2. *tail*: Pointer ini menunjuk ke elemen terakhir dari list. Tail berguna terutama ketika traversal dari belakang atau saat menambahkan elemen di akhir list.

- **Fungsi-fungsi yang Ada**

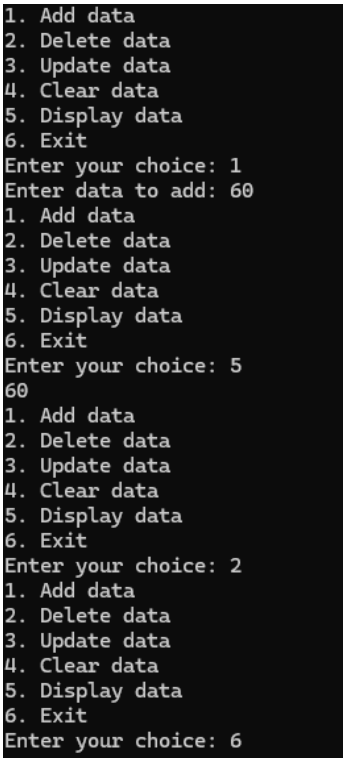
  1. *Insert* (Menambahkan Elemen di Depan List):
     Fungsi ini menambahkan node baru ke depan list. Jika list kosong, node baru akan menjadi elemen pertama dan satu-satunya, yang artinya *head* dan *tail* akan menunjuk ke node yang sama. Jika list tidak kosong, elemen baru akan disambungkan di depan *head* yang lama, dan *head* akan diperbarui untuk menunjuk ke node baru.
  2. *Delete* (Menghapus Elemen di Depan List):
     Fungsi ini menghapus elemen pertama dalam list. Jika list hanya memiliki satu elemen, setelah penghapusan, *head* dan *tail* akan diatur menjadi *nullptr*. Jika ada lebih dari satu elemen, fungsi ini memperbarui pointer *head* untuk menunjuk ke elemen berikutnya, dan pointer *prev* dari elemen baru tersebut diatur menjadi *nullptr*.
  3. *Update* (Memperbarui Data dalam List):
     Fungsi ini mencari elemen tertentu berdasarkan data yang diberikan. Jika elemen ditemukan, data dalam elemen tersebut akan diperbarui dengan data baru yang dimasukkan oleh pengguna. Jika elemen tidak ditemukan, program akan menginformasikan bahwa data tidak ada.
  4. *DeleteAll* (Menghapus Semua Elemen dalam List):
     Fungsi ini menghapus semua elemen dari list. Dalam proses ini, setiap node

akan dihapus satu per satu dengan memanfaatkan loop yang terus bergerak dari *head* hingga *tail*.

5. *Display* (Menampilkan Semua Elemen dalam List):
   Fungsi ini menampilkan semua elemen dari *head* hingga *tail*. Elemen akan ditampilkan secara berurutan sesuai dengan urutan penambahan.

- **Interaksi dengan Pengguna**
  Program menyediakan antarmuka berbasis teks (menu) yang memungkinkan pengguna memilih beberapa opsi:
  1. Menambahkan data: Pengguna dapat menambahkan elemen ke list.
  2. Menghapus elemen: Pengguna dapat menghapus elemen pertama dari list.
  3. Memperbarui data: Pengguna dapat memperbarui elemen tertentu dalam list.
  4. Menghapus semua data: Semua elemen dalam list akan dihapus.
  5. Menampilkan data: Program akan menampilkan semua elemen yang ada di dalam list.
  6. Keluar: Mengakhiri program.

- **Cara Kerja Program Secara Umum**
  1. Ketika program dijalankan, pengguna diberikan pilihan untuk memasukkan data ke dalam *Doubly Linked List*. Setiap kali pengguna menambahkan elemen, program menempatkan elemen tersebut di awal list dan mengatur ulang pointer *head* dan *prev* dari elemen sebelumnya.
  2. Jika pengguna ingin menghapus elemen, program akan menghapus elemen dari depan (*head*), memperbarui pointer, dan menghapus memori dari node yang dihapus.
  3. Pengguna juga bisa memperbarui elemen yang ada di dalam list dengan mencari elemen lama, lalu menggantinya dengan nilai baru.
  4. Fitur lain termasuk kemampuan untuk menghapus semua elemen sekaligus, yang akan mengosongkan list, dan fungsi untuk menampilkan seluruh elemen yang ada dalam *Doubly Linked List*.



IV. UNGUIDED

1. Task

Program ini mengimplementasikan struktur data *doubly linked list* untuk mengelola daftar kendaraan berdasarkan atribut seperti nomor polisi, warna, dan tahun pembuatan. Terdapat tiga file utama dalam kode ini: `doublelist.h`, `doublelist.cpp`, dan `main.cpp`, masing-masing memiliki fungsi dan tanggung jawab yang berbeda.

// coding doublelist.h //

```
// doublelist.h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H
#include <string>
using namespace std;

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

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);
bool isKendaraanExist(List L, string nopol);
address findElm(List L, string nopol);  // Task 2 function
void deleteFirst(List &L, address &P);  // Task 3 function
void deleteLast(List &L, address &P);   // Task 3 function
void deleteAfter(List &L, address Prec, address &P);  // Task 3 function

#endif
```

- **doublelist.h**

  Header file ini mendefinisikan struktur data dan fungsi yang akan digunakan dalam implementasi linked list. Berikut detail elemen-elemen di dalamnya:

  - Struct `infotype`: Struktur ini menyimpan informasi kendaraan berupa `nopol` (nomor polisi), `warna`, dan `thnBuat` (tahun pembuatan).
  - Pointer `address` dan `ElmList`: Pointer `address` didefinisikan sebagai penunjuk ke elemen `ElmList`. Struktur `ElmList` sendiri adalah elemen tunggal dari linked list yang menyimpan data `info` (dari tipe `infotype`), serta dua pointer `next` dan `prev` untuk menghubungkan elemen berikutnya dan sebelumnya.
  - Struct `List`: Menyimpan dua pointer, yaitu `First` dan `Last`, yang menunjuk pada elemen pertama dan terakhir dalam linked list. Ini memudahkan pengaksesan kedua ujung list tanpa harus melakukan traversing dari satu arah saja.

- Deklarasi Fungsi: Berisi deklarasi beberapa fungsi utama yang mencakup operasi dasar linked list, seperti membuat list (`CreateList`), mengalokasi node (`alokasi`), mencetak isi list (`printInfo`), serta menambahkan dan menghapus node. Ada juga fungsi untuk mencari node tertentu (`findElm`) dan mengecek keberadaan data berdasarkan nomor polisi (`isKendaraanExist`).

// coding doublelist.cpp //

```cpp
// doublelist.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;
}
void dealokasi(address &P) {
    delete P;
}

void printInfo(List L) {
    address P = L.First;
    cout << "\nDATA LIST 1" << endl;
    while (P != NULL) {
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna     : " << P->info.warna << endl;
        cout << "tahun     : " << P->info.thnBuat << 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;
    }
}

bool isKendaraanExist(List L, string nopol) {
    address P = L.First;
    while (P != NULL) {
        if (P->info.nopol == nopol) {
            return true;
        }
        P = P->next;
    }
    return false;
}

// Find element based on nomor polisi
address findElm(List L, string nopol) {
    address P = L.First;
    while (P != NULL) {
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return NULL;
}
```

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

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

// Delete after a specified element
void deleteAfter(List &L, address Prec, address &P) {
    if (Prec != NULL && Prec->next != NULL) {
        P = Prec->next;
        if (P == L.Last) {
            deleteLast(L, P);
        } else {
            Prec->next = P->next;
            P->next->prev = Prec;
            P->next = NULL;
            P->prev = NULL;
        }
    }
}
```

- **doublelist.cpp**
  File ini berfungsi sebagai implementasi dari deklarasi yang ada di `doublelist.h`. Berikut rincian dari fungsi-fungsi utama di dalamnya:
  - CreateList: Inisialisasi linked list dengan membuat `First` dan `Last` menjadi `NULL`, yang menandakan bahwa list masih kosong.
  - alokasi dan dealokasi: Fungsi `alokasi` digunakan untuk membuat elemen baru dalam list dengan data yang diberikan sebagai parameter, sementara `dealokasi` berfungsi untuk menghapus elemen dari memori.
  - printInfo: Mencetak seluruh isi list mulai dari elemen pertama hingga akhir. Untuk setiap elemen, fungsi ini menampilkan atribut `nopol`, `warna`, dan `thnBuat`.
  - insertLast: Menambahkan elemen baru di akhir list. Jika list kosong, elemen baru menjadi `First` dan `Last`. Jika tidak, elemen baru ditambahkan di akhir, dan pointer `Last` diperbarui.
  - isKendaraanExist: Mengecek apakah nomor polisi tertentu sudah ada dalam list. Fungsi ini akan mengembalikan nilai `true` jika ditemukan, dan `false` jika tidak.

- findElm: Mencari elemen berdasarkan `nopol`. Fungsi ini mengembalikan pointer ke elemen yang sesuai jika ditemukan, atau `NULL` jika tidak ada elemen yang cocok.
- deleteFirst, deleteLast, dan deleteAfter**: Ketiga fungsi ini menghapus elemen dari posisi tertentu di dalam list:
- deleteFirst menghapus elemen pertama.
- deleteLast menghapus elemen terakhir.
- deleteAfter menghapus elemen setelah elemen tertentu yang ditunjuk oleh parameter `Prec`. Jika elemen yang ingin dihapus adalah elemen terakhir, maka `deleteLast` akan dipanggil.

// coding main.cpp //

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

int main() {
    List L;
    infotype kendaraan;
    address P;

    CreateList(L);

    // Insert sample data
    while (true) {
        cout << "masukkan nomor polisi: ";
        cin >> kendaraan.nopol;
                    string infotype::nopol
        if (isKendaraanExist(L, kendaraan.nopol)) {
            cout << "nomor polisi sudah terdaftar" << endl;
            continue;
        }

        cout << "masukkan warna kendaraan: ";
        cin >> kendaraan.warna;
        cout << "masukkan tahun kendaraan: ";
        cin >> kendaraan.thnBuat;

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

        if (L.First != NULL && L.First->next != NULL && L.First->next->next != NULL) {
            break;
        }
    }

    // Display the list
    printInfo(L);
```

```cpp
    // Find element by nomor polisi
    cout << "\nMasukkan Nomor Polisi yang dicari: ";
    string nopolCari;
    cin >> nopolCari;
    P = findElm(L, nopolCari);
    if (P != NULL) {
        cout << "Nomor Polisi : " << P->info.nopol << endl;
        cout << "Warna        : " << P->info.warna << endl;
        cout << "Tahun        : " << P->info.thnBuat << endl;
    } else {
        cout << "Data dengan nomor polisi " << nopolCari << " tidak ditemukan." << endl;
    }

    // Delete element based on user input
    cout << "\nMasukkan Nomor Polisi yang ingin dihapus: ";
    string nopolHapus;
    cin >> nopolHapus;

    P = findElm(L, nopolHapus);
    if (P != NULL) {
        address deletedNode = NULL;

        if (P == L.First) {
            deleteFirst(L, deletedNode);
        } else if (P == L.Last) {
            deleteLast(L, deletedNode);
        } else {
            deleteAfter(L, P->prev, deletedNode);
        }

        dealokasi(deletedNode);
        cout << "Data dengan nomor polisi " << nopolHapus << " berhasil dihapus." << endl;
    } else {
        cout << "Data dengan nomor polisi " << nopolHapus << " tidak ditemukan, sehingga tidak dapat dihapus." << endl;
    }

    // Display the list after deletion
    printInfo(L);

    return 0;
}
```

File ini adalah bagian utama dari program yang menjalankan fungsi-fungsi dalam `doublelist`. Berikut adalah langkah-langkah utama dalam program ini:

- Inisialisasi dan Input Data: Program memulai dengan menginisialisasi list menggunakan `CreateList`. Pengguna kemudian diminta memasukkan data kendaraan berupa nomor polisi, warna, dan tahun pembuatan. Fungsi `isKendaraanExist` digunakan untuk memastikan bahwa nomor polisi yang dimasukkan belum terdaftar sebelumnya.
- Insert Data: Data yang berhasil diverifikasi kemudian dimasukkan ke dalam list menggunakan `insertLast`.
- Display List: Data dalam list ditampilkan menggunakan `printInfo` setelah beberapa elemen dimasukkan.
- Mencari Elemen: Pengguna dapat mencari data kendaraan berdasarkan nomor polisi. Fungsi `findElm` digunakan untuk menemukan elemen dengan nomor polisi yang dicari. Jika ditemukan, data kendaraan tersebut akan ditampilkan.
- Menghapus Elemen: Pengguna dapat menghapus elemen dengan memasukkan nomor polisi kendaraan yang diinginkan. Jika ditemukan, elemen tersebut dihapus dengan fungsi yang sesuai (`deleteFirst`, `deleteLast`, atau `deleteAfter`). Setelah penghapusan, data dihapus dari memori menggunakan `dealokasi`.
- Tampilkan List Setelah Penghapusan: Setelah menghapus data, program kembali menampilkan isi list untuk memverifikasi perubahan.

Program ini mengilustrasikan cara penggunaan *doubly linked list* dalam mengelola data kendaraan dengan berbagai operasi dasar, seperti penambahan, penghapusan, pencarian, dan pengecekan data.

Hasil Output :

```
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: 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
```

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

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

DATA LIST 1
no polisi : D001
warna     : hitam
tahun     : 90
no polisi : D004
warna     : kuning
tahun     : 90
```

V. KESIMPULAN

Kesimpulan dari implementasi ADT Double Linked List pada proyek ini adalah bahwa struktur data double linked list dapat digunakan secara efektif untuk menyimpan dan mengelola data yang memiliki keterkaitan dua arah, seperti data kendaraan dalam sistem manajemen sederhana. Proses penambahan data dapat dilakukan dengan mudah melalui prosedur `insertLast`, sementara pencarian data menjadi efisien dengan adanya fungsi `findElm`. Selain itu, penghapusan data dapat dilakukan sesuai posisi elemen melalui prosedur `deleteFirst`, `deleteLast`, dan `deleteAfter`, yang memungkinkan pengelolaan data dalam list secara fleksibel. Penggunaan prosedur-prosedur tersebut mempermudah pemrogram dalam menambah, mencari, dan menghapus elemen di berbagai posisi dalam list, sesuai dengan kebutuhan program. Implementasi ini menunjukkan pentingnya pengelolaan memori dan pointer yang baik dalam operasi linked list, serta manfaat struktur data ini dalam aplikasi yang memerlukan pengelolaan data dinamis.