

**LAPORAN PRAKTIKUM**  
**Modul VI**  
**“Double Linked List”**



**Disusun Oleh:**  
**Zivana Afra Yulianto -2211104039**  
**SE-07-02**

**Dosen:**  
**Wahyu Andi Saputra**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

## 1. Tujuan

Tujuan dari praktikum ini adalah untuk memahami dan mengimplementasikan struktur data **Double Linked List (DLL)** dalam bahasa C++. Praktikum ini mencakup pembuatan, penambahan, penghapusan, pencarian, dan penampilan elemen dalam doubly linked list. Selain itu, dilakukan juga pembaruan dan penghapusan data kendaraan berdasarkan nomor polisi.

## 2. Landasan Teori

- **Linked List**

Linked list adalah struktur data linear yang terdiri dari node-node yang saling terhubung melalui pointer. Setiap node menyimpan data dan pointer ke node berikutnya (dan dalam doubly linked list, juga menyimpan pointer ke node sebelumnya).

- **Doubly Linked List (DLL)**

Doubly linked list adalah jenis linked list di mana setiap node memiliki dua pointer, satu menunjuk ke node berikutnya dan satu lagi menunjuk ke node sebelumnya. Hal ini memungkinkan traversal baik maju maupun mundur di dalam list. DLL lebih fleksibel dibandingkan singly linked list tetapi membutuhkan lebih banyak memori karena tambahan pointer.

- **Operasi Dasar pada DLL**

- **Inisialisasi List:** Membuat list kosong dengan `head` dan `tail` menunjuk ke `nullptr`.
- **Insert Node:** Menambahkan elemen pada awal atau akhir list. Dalam praktikum ini, insert dilakukan di akhir list.
- **Delete Node:** Menghapus elemen dari awal, akhir, atau setelah node tertentu dalam list.
- **Search Node:** Mencari node berdasarkan kriteria tertentu, seperti nomor polisi kendaraan.
- **Display Node:** Menampilkan semua elemen dalam list.

- **Pointer dan Dynamic Memory Allocation**

Pointer sangat penting dalam linked list karena memungkinkan elemen-elemen dalam list untuk diakses dan dimanipulasi secara dinamis. Penggunaan `new` dan `delete` dalam C++ memungkinkan pengelolaan memori secara dinamis.

### 3. Guided

Code :

```
#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;
        }
        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;

        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;
            }
        }
    }
    return 0;
}

```

Sreenshoot Output :

```
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: 12
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
12
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS C:\STD_Zivana_Afra_Yulianto>
```

Deskripsi :

Kode ini mengimplementasikan Doubly Linked List di C++, di mana setiap elemen atau node memiliki pointer ke elemen sebelumnya dan berikutnya. Berikut adalah fungsi utama dalam kode ini:

1. Node Class: Menyimpan data integer dan pointer `prev` serta `next` ke elemen sebelum dan sesudahnya dalam list.
2. DoublyLinkedList Class:
  - `insert(int data)`: Menambahkan elemen baru di awal list. Jika list kosong, elemen juga menjadi `tail`.
  - `deleteNode()`: Menghapus elemen pertama (`head`) dari list. Jika hanya satu elemen, `tail` diatur ke `nullptr`.
  - `update(int oldData, int newData)`: Mengubah elemen bernilai `oldData` menjadi `newData`, mengembalikan `true` jika berhasil, atau `false` jika tidak ditemukan.
  - `deleteAll()`: Menghapus semua elemen dalam list dan mengosongkannya.
  - `display()`: Menampilkan semua elemen dari `head` ke `tail`.
3. Main Function:
  - Menyediakan menu interaktif untuk menambah, menghapus, memperbaiki, menghapus semua, dan menampilkan elemen list, serta keluar dari program.

Contoh: Jika pengguna menambah data 10, 20, lalu 30, list akan berisi 30 -> 20 -> 10.

## 4. Unguided

### 1. Code :

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

// Type infotype
struct kendaraan
{
    string nopol;
    string warna;
    int thnBuat;
};

// Type address sebagai pointer to ElmList
typedef struct ElmList *address;

// Type ElmList
struct ElmList
{
    kendaraan info;
    address next;
    address prev;
};

// Type List
struct List
{
    address First;
    address Last;
};

// Prosedur dan Fungsi ADT
void CreateList(List &L);
address alokasi(kendaraan 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

// Prosedur CreateList
void CreateList(List &L)
{
    L.First = nullptr;
    L.Last = nullptr;
}

// Fungsi alokasi
address alokasi(kendaraan x)
{
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

// Prosedur dealokasi
void dealokasi(address &P)
{
    delete P;
    P = nullptr;
}

// Prosedur untuk menampilkan semua elemen dalam list
void printInfo(List L)
{
    address P = L.First;
    while (P != nullptr)
    {
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna      : " << P->info.warna << endl;
        cout << "tahun       : " << P->info.thnBuat << endl;
        cout << "-----" << endl;
        P = P->next;
    }
}
```

```

// Prosedur untuk menambahkan elemen di akhir list
void insertLast(List &L, address P)
{
    if (L.First == nullptr)
    { // List kosong
        L.First = P;
        L.Last = P;
    }
    else
    {
        L.Last->next = P;
        P->prev = L.Last;
        L.Last = P;
    }
}

// Fungsi untuk mencari elemen berdasarkan nopol
address findElm(List L, string nopol)
{
    address P = L.First;
    while (P != nullptr)
    {
        if (P->info.nopol == nopol)
        {
            return P;
        }
        P = P->next;
    }
    return nullptr; // Tidak ditemukan
}

// Prosedur untuk menghapus elemen pertama
void deleteFirst(List &L, address &P)
{
    if (L.First != nullptr)
    {
        P = L.First;
        if (L.First == L.Last)
        { // Hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        }
        else
        {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        P->next = nullptr;
        dealokasi(P);
    }
}

// Prosedur untuk menghapus elemen terakhir
void deleteLast(List &L, address &P)
{
    if (L.Last != nullptr)
    {
        P = L.Last;
        if (L.First == L.Last)
        { // Hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        }
        else
        {
            L.Last = L.Last->prev;
            L.Last->next = nullptr;
        }
        P->prev = nullptr;
        dealokasi(P);
    }
}

// Prosedur untuk menghapus elemen setelah Prec
void deleteAfter(address Prec, address &P)
{
    if (Prec != nullptr && Prec->next != nullptr)
    {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != nullptr)
        {
            P->next->prev = Prec;
        }
        else
        {
            Prec->prev = nullptr;
        }
        P->next = nullptr;
        P->prev = nullptr;
        dealokasi(P);
    }
}

```

```

bool isNopolExist(List L, string nopol)
{
    return findElm(L, nopol) != nullptr;
}

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

    int choice;
    do
    {
        cout << "\nMENU:\n";
        cout << "1. Tambah data kendaraan\n";
        cout << "2. Cari data kendaraan berdasarkan nomor polisi\n";
        cout << "3. Tampilkan seluruh data kendaraan\n";
        cout << "4. Hapus data kendaraan\n";
        cout << "5. Keluar\n";
        cout << "Masukkan pilihan: ";
        cin >> choice;

        if (choice == 1)
        {
            kendaraan k;
            cout << "Masukkan nomor polisi: ";
            cin >> k.nopol;
            if (isNopolExist(L, k.nopol))
            {
                cout << "Nomor polisi sudah terdaftar\n";
                continue;
            }
            cout << "Masukkan warna kendaraan: ";
            cin >> k.warna;
            cout << "Masukkan tahun kendaraan: ";
            cin >> k.thnBuat;

            insertLast(L, alokasi(k));
        }
        else if (choice == 2)
        {
            string nopol;
            cout << "Masukkan nomor polisi yang dicari: ";
            cin >> nopol;
            address found = findElm(L, nopol);
            if (found)
            {
                cout << "\nData ditemukan:\n";
                cout << "No Polisi : " << found->info.nopol << endl;
                cout << "Warna      : " << found->info.warna << endl;
                cout << "Tahun       : " << found->info.thnBuat << endl;
            }
            else
            {
                cout << "Data tidak ditemukan\n";
            }
        }
        else if (choice == 3)
        {
            cout << "\nDATA LIST:\n";
            printInfo(L);
        }
        else if (choice == 4)
        {
            string nopol;
            cout << "Masukkan nomor polisi yang ingin dihapus: ";
            cin >> nopol;
            address P = findElm(L, nopol);

            if (P != nullptr)
            {
                if (P == L.First)
                {
                    deleteFirst(L, P);
                }
                else if (P == L.Last)
                {
                    deleteLast(L, P);
                }
                else
                {
                    deleteAfter(P->prev, P);
                }
                cout << "Data berhasil dihapus\n";
            }
            else
            {
                cout << "Data tidak ditemukan\n";
            }
        }
    } while (choice != 5);

    return 0;
}

```



Screenshoot Output :

```
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 1
Masukkan nomor polisi: D003
Masukkan warna kendaraan: putih
Masukkan tahun kendaraan: 2024

MENU:
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 3

DATA LIST:
no polisi : D001
warna      : hitam
tahun      : 2023
-----
no polisi : D003
warna      : putih
tahun      : 2024
-----
```

Deskripsi :

Program ini mengelola data kendaraan menggunakan struktur Doubly Linked List. Program memiliki beberapa fitur untuk menambah, mencari, menampilkan, dan menghapus data kendaraan berdasarkan nomor polisi. Setiap kendaraan memiliki atribut nomor polisi, warna, dan tahun pembuatan. Berikut adalah fungsi utama dari program ini:

1. Tambah Data Kendaraan: Menambahkan data kendaraan baru ke dalam list. Program memeriksa apakah nomor polisi sudah terdaftar sebelumnya untuk menghindari duplikasi.
2. Cari Data Kendaraan: Mencari data kendaraan dalam list berdasarkan nomor polisi yang diberikan oleh pengguna. Jika ditemukan, program menampilkan informasi kendaraan tersebut.
3. Tampilkan Seluruh Data Kendaraan: Menampilkan semua data kendaraan yang ada dalam list dengan format yang rapi.
4. Hapus Data Kendaraan: Menghapus data kendaraan dari list berdasarkan nomor polisi. Program memastikan untuk menghapus elemen dari awal, tengah, atau akhir list sesuai posisi data yang ingin dihapus.
5. Keluar: Mengakhiri program.

Program ini memanfaatkan beberapa prosedur dan fungsi, seperti `insertLast` untuk menambahkan data di akhir list, `findElm` untuk mencari data berdasarkan nomor polisi, dan `deleteFirst`, `deleteLast`, serta `deleteAfter` untuk menghapus elemen dari list sesuai posisi data yang ingin dihapus.

2. Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru. fungsi  
findElm( L : List, x : infotype ) : address

Code :

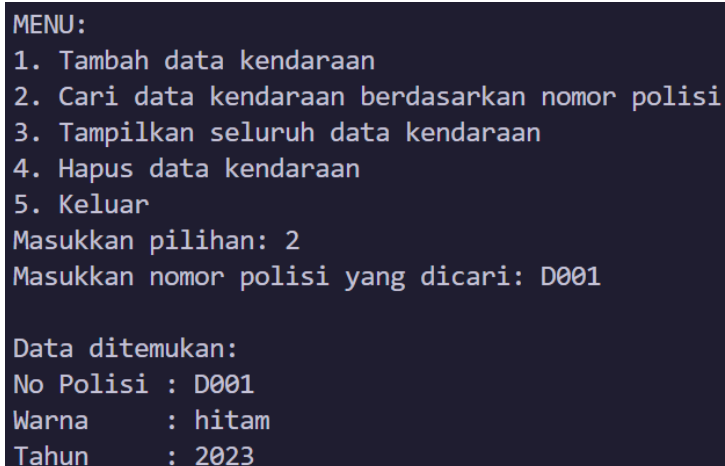
```
// Fungsi untuk mencari elemen berdasarkan nopol
address findElm(List L, string nopol)
{
    address P = L.First;
    while (P != nullptr)
    {
        if (P->info.nopol == nopol)
        {
            return P;
        }
        P = P->next;
    }
    return nullptr; // Tidak ditemukan
}
```

Code diatas merupakan code logika untuk mencari elemen berdasarkan nopol yang dideklarasikan dengan nama findElm(List L, String nopol).

Kemudian melakukan pemanggilan pada main dengan code

```
else if (choice == 2)
{
    string nopol;
    cout << "Masukkan nomor polisi yang dicari: ";
    cin >> nopol;
    address found = findElm(L, nopol);
    if (found)
    {
        cout << "\nData ditemukan:\n";
        cout << "No Polisi : " << found->info.nopol << endl;
        cout << "Warna      : " << found->info.warna << endl;
        cout << "Tahun       : " << found->info.thnBuat << endl;
    }
    else
    {
        cout << "Data tidak ditemukan\n";
    }
}
```

Screenshoot Output :



```
MENU:
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 2
Masukkan nomor polisi yang dicari: D001

Data ditemukan:
No Polisi : D001
Warna      : hitam
Tahun       : 2023
```

Deskripsi :

Kode di atas adalah logika untuk mencari elemen dalam Doubly Linked List berdasarkan nomor polisi (nopol). Fungsi ini diberi nama `findElm(List L, string nopol)`, di mana parameter `L` adalah list yang akan dicari, dan `nopol` adalah nomor polisi yang dicari.

1. Logika Pencarian:

- Fungsi `findElm` menggunakan pointer `P` yang diawali dari elemen pertama list (`L.First`).
- Dalam loop `while`, kode akan memeriksa setiap elemen dalam list hingga `P` bernilai `nullptr`, yang menandakan akhir dari list.
- Jika ditemukan elemen dengan `nopol` yang sama dengan input pengguna, fungsi akan langsung mengembalikan alamat elemen tersebut (`return P`).
- Jika elemen dengan `nopol` yang dicari tidak ditemukan hingga akhir list, maka fungsi mengembalikan `nullptr`.

2. Penggunaan Fungsi di Main:

- Pada bagian `main`, terdapat menu pilihan untuk mencari data kendaraan berdasarkan nomor polisi (`choice == 2`).
- Pengguna diminta memasukkan nomor polisi yang dicari (`cin >> nopol`).
- Fungsi `findElm` kemudian dipanggil, dan hasilnya disimpan dalam variabel `found`.
- Jika `found` tidak `nullptr`, artinya data ditemukan. Informasi kendaraan (nomor polisi, warna, tahun) akan ditampilkan ke pengguna.
- Jika `found` bernilai `nullptr`, program menampilkan pesan bahwa data tidak ditemukan.

Kode ini membantu pengguna untuk mencari dan melihat informasi kendaraan dengan mudah berdasarkan nomor polisi yang dicari.

3. Hapus elemen dengan nomor polisi D003 dengan prosedur delete.

- prosedur deleteFirst( in/out L : List, in/out P : address )
- prosedur deleteLast( in/out L : List, in/out P : address )
- prosedur deleteAfter( in Prec : address, in/out: P : address )

Code logika :

```
// Prosedur untuk menghapus elemen pertama
void deleteFirst(List &L, address &P)
{
    if (L.First != nullptr)
    {
        P = L.First;
        if (L.First == L.Last)
        { // Hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        }
        else
        {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        P->next = nullptr;
        dealokasi(P);
    }
}

// Prosedur untuk menghapus elemen terakhir
void deleteLast(List &L, address &P)
{
    if (L.Last != nullptr)
    {
        P = L.Last;
        if (L.First == L.Last)
        { // Hanya satu elemen
            L.First = nullptr;
            L.Last = nullptr;
        }
        else
        {
            L.Last = L.Last->prev;
            L.Last->next = nullptr;
        }
        P->prev = nullptr;
        dealokasi(P);
    }
}

// Prosedur untuk menghapus elemen setelah Prec
void deleteAfter(address Prec, address &P)
{
    if (Prec != nullptr && Prec->next != nullptr)
    {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != nullptr)
        {
            P->next->prev = Prec;
        }
        else
        {
            Prec->prev = nullptr;
        }
        P->next = nullptr;
        P->prev = nullptr;
        dealokasi(P);
    }
}
```

### Pemanggilan pada main :

```
else if (choice == 4)
{
    string nopol;
    cout << "Masukkan nomor polisi yang ingin dihapus: ";
    cin >> nopol;
    address P = findElm(L, nopol);

    if (P != nullptr)
    {
        if (P == L.First)
        {
            deleteFirst(L, P);
        }
        else if (P == L.Last)
        {
            deleteLast(L, P);
        }
        else
        {
            deleteAfter(P->prev, P);
        }
        cout << "Data berhasil dihapus\n";
    }
    else
    {
        cout << "Data tidak ditemukan\n";
    }
}
```

### Screenshoot Output :

```
MENU:
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 3

DATA LIST:
no polisi : D001
warna     : hitam
tahun     : 2023
-----
no polisi : D003
warna     : putih
tahun     : 2024
-----
```

(sebelum penghapusan data)

```
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 4
Masukkan nomor polisi yang ingin dihapus: D003
Data berhasil dihapus

MENU:
1. Tambah data kendaraan
2. Cari data kendaraan berdasarkan nomor polisi
3. Tampilkan seluruh data kendaraan
4. Hapus data kendaraan
5. Keluar
Masukkan pilihan: 3

DATA LIST:
no polisi : D001
warna     : hitam
tahun     : 2023
-----
```

(setelah penghapusan data)

### Deskripsi :

1. **deleteFirst**: Menghapus elemen pertama. Jika list hanya memiliki satu elemen, `First` dan `Last` diatur ke `nullptr`. Jika lebih dari satu, elemen pertama berpindah ke elemen berikutnya.
2. **deleteLast**: Menghapus elemen terakhir. Jika list hanya memiliki satu elemen, `First` dan `Last` diatur ke `nullptr`. Jika lebih dari satu, elemen terakhir berpindah ke elemen sebelumnya.
3. **deleteAfter**: Menghapus elemen setelah elemen tertentu (`Prec`). Menghubungkan `Prec` dengan elemen setelah yang dihapus.
4. **Pemanggilan di main**: Program meminta nomor polisi, mencari elemen tersebut, dan menghapusnya dengan prosedur yang sesuai

## 5. Kesimpulan

Praktikum ini berhasil mengimplementasikan doubly linked list dengan operasi dasar seperti penambahan, penghapusan, pencarian, dan tampilan elemen. Melalui praktikum ini, diperoleh pemahaman tentang bagaimana menggunakan pointer untuk mengelola elemen-elemen dalam struktur linked list, serta bagaimana mengelola memori dinamis dengan tepat. Penggunaan doubly linked list memberikan fleksibilitas dalam traversal list, baik dari depan maupun belakang. Struktur ini sangat berguna untuk aplikasi yang membutuhkan kemudahan akses data dari kedua arah.