# LAPORAN PRAKTIKUM
# Modul 6
# Double Linked List (Bagian Pertama)


**Disusun Oleh:**
**Berlian Seva Astryana - 2311104067**
**Kelas:**
**SE-07-02**

**Dosen :**
**Wahyu Andi Saputra, S.Pd., M.Eng**

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

**2024**

1. **Tujuan**
   1.1 Memahami konsep modul linked list.
   1.2 Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C
2. **Landasan Teori**

Double Linked List (DLL) adalah struktur data dinamis yang terdiri dari kumpulan elemen di mana setiap elemen memiliki dua pointer: satu menunjuk ke elemen berikutnya dan satu lagi menunjuk ke elemen sebelumnya. Dalam DLL, elemen pertama memiliki pointer yang menunjuk ke elemen berikutnya dan pointer prev bernilai NULL, sedangkan elemen terakhir memiliki pointer next yang bernilai NULL. Struktur ini memungkinkan akses yang fleksibel karena elemen dapat dijelajahi baik maju (menggunakan pointer next) maupun mundur (menggunakan pointer prev). Perbedaan mendasar dengan *Single Linked List* adalah bahwa DLL memiliki dua pointer ini, sehingga mempermudah manipulasi data, terutama ketika menghapus atau menyisipkan elemen di posisi tertentu.

Pada Double Linked List, terdapat dua pointer utama, yaitu First yang menunjuk ke elemen pertama dalam list dan Last yang menunjuk ke elemen terakhir. Hal ini memungkinkan akses lebih cepat ke elemen di awal maupun di akhir list. Sebagai contoh, dalam aplikasi penyimpanan data kendaraan, First dapat digunakan untuk menunjuk pada data kendaraan yang pertama kali masuk dalam list, sedangkan Last menunjuk pada data kendaraan terakhir. Setiap elemen dalam list dapat menyimpan informasi spesifik, seperti nomor polisi (nopol), warna, dan tahun pembuatan kendaraan.

Beberapa operasi dasar pada DLL meliputi penyisipan dan penghapusan elemen. Proses *insert first* dilakukan dengan menyisipkan elemen baru di awal list, di mana pointer next elemen baru akan menunjuk ke elemen pertama saat ini, sementara pointer prev elemen pertama diperbarui untuk menunjuk ke elemen baru. Setelah itu, pointer First diperbarui untuk menunjuk ke elemen baru tersebut. Sementara itu, operasi *insert last* dilakukan dengan menambahkan elemen di akhir list. Pointer next elemen terakhir diperbarui agar menunjuk ke elemen baru, sementara elemen baru menunjuk ke elemen sebelumnya melalui pointer prev. Pointer Last juga diperbarui untuk menunjuk ke elemen baru yang dimasukkan.

Selain penyisipan, Double Linked List juga memungkinkan penghapusan elemen, baik di awal, akhir, maupun setelah elemen tertentu. Proses *delete first* dilakukan dengan memperbarui pointer First untuk menunjuk ke elemen setelah elemen pertama yang dihapus. Jika list hanya memiliki satu elemen, maka pointer First dan Last diatur ke NULL. Pada proses *delete last*, pointer Last diubah untuk menunjuk ke elemen sebelum elemen terakhir yang dihapus. Apabila hanya ada satu elemen, pointer First dan Last akan diatur menjadi NULL. Penghapusan elemen setelah elemen tertentu, yang dikenal dengan *delete after*, dilakukan dengan memperbarui pointer next elemen yang ada di posisi Prec agar menunjuk ke elemen setelah elemen yang dihapus.

Dalam penerapan Double Linked List untuk studi kasus manajemen data kendaraan, setiap elemen dapat menyimpan informasi kendaraan seperti nomor polisi (nopol), warna, dan tahun pembuatan. Penambahan data kendaraan baru pada akhir list dapat

dilakukan menggunakan prosedur insert last, dengan catatan bahwa nomor polisi yang diinput belum ada di dalam list. Proses pencarian data kendaraan berdasarkan nomor polisi tertentu dapat dilakukan dengan fungsi pencarian (findElm), dan hasilnya adalah alamat elemen yang memuat informasi kendaraan tersebut, atau NULL jika nomor polisi tidak ditemukan. Jika perlu menghapus elemen, prosedur delete akan digunakan, misalnya untuk menghapus kendaraan tertentu berdasarkan nomor polisi.

Dengan keunggulan berupa akses dua arah, Double Linked List menjadi struktur data yang efisien untuk aplikasi dinamis. Hal ini sangat berguna ketika data memerlukan pengaksesan yang cepat, baik dari awal maupun dari akhir, serta memudahkan proses penghapusan dan penyisipan elemen di berbagai posisi.

## 3. Guided

Telkom University Purwokerto

```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;
        } 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;
}
```

Contoh output:

```
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 14
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 12
Enter new data: 17
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
14 17
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
```

Penjelasan:

Program di atas merupakan implementasi dari struktur data Doubly Linked List yang memungkinkan pengguna untuk menambah, menghapus, memperbarui, menghapus semua, dan menampilkan elemen dari list tersebut. Kelas DoublyLinkedList memiliki dua pointer: head (untuk menunjuk elemen pertama) dan tail (untuk menunjuk elemen terakhir). Fungsi insert() menambahkan elemen baru di depan list, sementara deleteNode() menghapus elemen pertama. Fungsi update() mencari data lama (oldData) dan menggantinya dengan data baru (newData) jika ditemukan. Fungsi deleteAll() menghapus semua elemen dalam list, dan display() menampilkan seluruh data dalam list. Di main(), terdapat menu interaktif yang memungkinkan pengguna untuk memilih operasi yang akan dilakukan. Program berjalan dalam loop hingga pengguna memilih opsi "Exit".

4. **Unguided**

Code program doublelist.h:

Telkom University Purwokerto

doublelist.h

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

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

typedef kendaraan infotype;
struct ElmList {
    infotype info;
    ElmList *next;
    ElmList *prev;
};
typedef ElmList* address;

struct List {
    address First;
    address Last;
};

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

#endif
```

Code program doublelist.cpp:

doublelist.cpp

```cpp
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>
using namespace std;

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

typedef kendaraan infotype;
struct ElmList {
    infotype info;
    ElmList *next;
    ElmList *prev;
};
typedef ElmList* address;

struct List {
    address First;
    address Last;
};

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

#endif
```

Telkom University Purwokerto

Code program main.cpp:

main.cpp

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

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

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

    string nopol, warna;
    int thnBuat;

    // Memasukkan beberapa data kendaraan
    for (int i = 0; i < 4; i++) {
        cout << "Masukkan nomor polisi: ";
        cin >> nopol;

        if (isNopolExists(L, nopol)) {
            cout << "Nomor polisi sudah terdaftar\n";
            continue;
        }

        cout << "Masukkan warna kendaraan: ";
        cin >> warna;
        cout << "Masukkan tahun kendaraan: ";
        cin >> thnBuat;

        infotype kendaraanBaru = {nopol, warna, thnBuat};
        address P = alokasi(kendaraanBaru);
        insertLast(L, P);
    }

    // Mencetak semua data di list
    printInfo(L);

    // Mencari elemen dengan nomor polisi tertentu
    cout << "Masukkan Nomor Polisi yang dicari: ";
    cin >> nopol;
    address hasilPencarian = findElm(L, nopol);

    if (hasilPencarian != nullptr) {
        cout << "\nNomor Polisi : " << hasilPencarian->info.nopol << "\n";
        cout << "Warna        : " << hasilPencarian->info.warna << "\n";
        cout << "Tahun        : " << hasilPencarian->info.thnBuat << "\n";
    } else {
        cout << "Nomor polisi tidak ditemukan\n";
    }

    // Menghapus elemen berdasarkan nomor polisi
    cout << "\nMasukkan Nomor Polisi yang akan 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);
        }
        dealokasi(P);
        cout << "Data dengan nomor polisi " << nopol << " berhasil dihapus.\n";
    } else {
        cout << "Nomor polisi tidak ditemukan.\n";
    }

    // Mencetak ulang semua data setelah penghapusan
    printInfo(L);

    return 0;
}
```

Penjelasan:

a. Membuat ADT Double Linked List di File “doublelist.h”: Kode ini mendefinisikan ADT untuk double linked list yang digunakan untuk menyimpan data kendaraan. Pertama, tipe infotype didefinisikan sebagai struct kendaraan, yang berisi atribut nopol untuk nomor polisi, warna untuk warna kendaraan, dan thnBuat sebagai tahun pembuatan kendaraan. Kemudian, address dideklarasikan sebagai tipe alias untuk pointer yang menunjuk ke ElmList, memudahkan penanganan pointer dalam operasi linked list. Struktur ElmList menjadi node dasar untuk list, menyimpan data kendaraan dalam info, serta pointer next untuk node berikutnya dan prev untuk node sebelumnya. Struktur List kemudian didefinisikan sebagai wadah utama linked list, yang memiliki pointer First untuk elemen pertama dan Last untuk elemen terakhir dalam list. Selain itu, terdapat prototipe beberapa prosedur dan fungsi untuk mengelola list ini, seperti CreateList untuk inisialisasi, alokasi untuk mengalokasikan memori node baru, dealokasi untuk membebaskan memori, printInfo untuk mencetak seluruh isi list, dan insertLast untuk menambahkan node di akhir list.
b. Implementasi ADT Double Linked List pada File “doublelist.cpp”: Kode implementasi ADT double linked list berada pada file doublelist.cpp. Prosedur CreateList menginisialisasi linked list dengan mengatur First dan Last ke nullptr, sehingga list kosong saat pertama kali dibuat. Fungsi alokasi membuat node baru yang memuat data kendaraan dari parameter x, mengalokasikan memori untuk ElmList, lalu mengatur next dan prev node baru tersebut sebagai nullptr. Fungsi dealokasi bertugas untuk melepaskan memori node P dan mengaturnya kembali menjadi nullptr setelah dihapus. Prosedur printInfo menampilkan seluruh data kendaraan dalam list dengan menelusuri list dari node First hingga node Last. Prosedur insertLast digunakan untuk menambahkan node baru di akhir list. Jika list kosong, node baru menjadi elemen pertama dan terakhir sekaligus. Jika tidak, node tersebut dihubungkan ke Last sebagai node terakhir baru.
c. Mencari Elemen dengan Nomor Polisi Tertentu: Fungsi findElm menerima dua parameter, yaitu List dan nopol. Fungsi ini digunakan untuk mencari kendaraan tertentu dalam list berdasarkan nomor polisinya (nopol). Fungsi ini menelusuri setiap node dalam list mulai dari First dan membandingkan nopol dalam data kendaraan dengan nopol yang dicari. Jika ditemukan kecocokan, fungsi akan mengembalikan pointer node tersebut; jika tidak ditemukan, fungsi mengembalikan nullptr. Fungsi ini memungkinkan pencarian elemen dalam list berdasarkan kriteria tertentu.
d. Menghapus Elemen Berdasarkan Nomor Polisi: Terdapat beberapa prosedur yang digunakan untuk menghapus elemen dalam list. Prosedur deleteFirst digunakan untuk menghapus elemen pertama (First) dalam list. Jika list hanya memiliki satu elemen, First dan Last diatur kembali menjadi nullptr, sehingga list menjadi kosong. Jika list memiliki lebih dari satu elemen, First diperbarui ke elemen berikutnya, dan pointer prev pada node baru First diatur ke nullptr. Prosedur deleteLast berfungsi untuk menghapus elemen terakhir (Last) dalam list. Jika hanya ada satu elemen, list

menjadi kosong. Jika terdapat lebih dari satu elemen, pointer Last diperbarui ke elemen sebelumnya, dan pointer next pada node baru Last diatur ke nullptr. Prosedur deleteAfter digunakan untuk menghapus node setelah node tertentu (Prec). Ini biasanya digunakan untuk menghapus elemen di tengah list. Node setelah Prec akan dihapus, dan Prec dihubungkan ke node yang ada setelah node yang dihapus, jika ada. Prosedur ini memastikan integritas list tetap terjaga setelah penghapusan elemen di tengah.

e. Uji Implementasi di "main.cpp": Di file main.cpp, ADT yang telah dibuat dan diimplementasikan diuji dalam beberapa operasi. Pertama, beberapa data kendaraan dimasukkan melalui input pengguna, di mana sebelum menambah data baru, nomor polisi dicek keberadaannya agar tidak terjadi duplikasi. Seluruh data kendaraan kemudian ditampilkan menggunakan prosedur printInfo untuk memastikan data telah ditambahkan dengan benar. Selanjutnya, fungsi findElm digunakan untuk mencari kendaraan berdasarkan nopol yang diinputkan pengguna, dan hasil pencarian ditampilkan ke layar. Pengguna juga dapat menghapus data kendaraan berdasarkan nopol tertentu. Setelah penghapusan, list diperbarui, dan data kendaraan dicetak ulang untuk memastikan elemen dengan nomor polisi tertentu berhasil dihapus dari list.

## 5. Kesimpulan

Double Linked List (Bagian Pertama)" memperkenalkan konsep dasar dan implementasi double linked list menggunakan bahasa C. Double linked list merupakan struktur data yang memungkinkan akses dua arah melalui dua pointer, next untuk elemen berikutnya dan prev untuk elemen sebelumnya. Modul ini mencakup operasi dasar, seperti menambahkan elemen di awal, akhir, atau tengah list, serta menghapus elemen di berbagai posisi. Praktikum ini bertujuan untuk membantu mahasiswa memahami implementasi ADT double linked list serta mengaplikasikan fungsi-fungsi penting untuk manipulasi data, seperti CreateList, insertLast, findElm, dan delete. Dengan menguasai double linked list, mahasiswa dapat mengelola data yang memerlukan akses dinamis dan fleksibel secara lebih efisien.