LAPORAN PRAKTIKUM
MODUL 6
DOUBLE LINKED LIST BAGIAN PERTAMA

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan

1. Memahami konsep modul linked list.
2. Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C ++.

B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini

2.1 Double Linked List

Double Linked List merupakan jenis struktur data yang setiap elemennya memiliki dua pointer: satu yang menunjuk pada elemen sebelumnya (*prev*) dan satu lagi yang menunjuk pada elemen berikutnya (*next*). Berbeda dengan Single Linked List, Double Linked List memudahkan traversing atau penelusuran dalam kedua arah, baik maju maupun mundur. Struktur ini terdiri dari beberapa komponen utama:

1. **First**: Pointer yang menunjuk pada elemen pertama dalam list.
2. **Last**: Pointer yang menunjuk pada elemen terakhir dalam list.
3. **Next**: Pointer dalam setiap elemen yang menunjuk ke elemen sesudahnya.
4. **Prev**: Pointer dalam setiap elemen yang menunjuk ke elemen sebelumnya.

Double Linked List memberikan fleksibilitas lebih tinggi dalam akses dan modifikasi data karena setiap elemen dapat diakses dari dua arah.

2.1.1 Insert

Operasi *insert* pada Double Linked List memungkinkan penyisipan elemen baru ke dalam list. Proses ini terbagi dalam beberapa metode:

- **Insert First**: Menambahkan elemen baru di awal list. Jika list kosong, elemen yang ditambahkan menjadi elemen pertama dan terakhir. Jika tidak, elemen baru ini akan dihubungkan sebagai

elemen pertama, sedangkan elemen sebelumnya menjadi elemen kedua.

- **Insert Last**: Menambahkan elemen baru di akhir list. Jika list kosong, elemen tersebut menjadi elemen pertama dan terakhir. Jika list tidak kosong, elemen baru dihubungkan sebagai elemen terakhir dan elemen sebelumnya tetap dihubungkan dengan *next* dan *prev*.
- **Insert After dan Insert Before**: *Insert After* menyisipkan elemen setelah elemen tertentu dalam list, sedangkan *Insert Before* menyisipkan elemen sebelum elemen tertentu. Kedua metode ini memungkinkan penyisipan elemen di posisi yang diinginkan berdasarkan elemen yang sudah ada.

2.1.2 Delete

Operasi *delete* dalam Double Linked List memungkinkan penghapusan elemen yang berada di awal, akhir, atau di antara dua elemen tertentu:

- **Delete First**: Menghapus elemen pertama pada list. Jika list hanya berisi satu elemen, baik *first* maupun *last* diatur menjadi Nil.
- **Delete Last**: Menghapus elemen terakhir dalam list. Mirip dengan *delete first*, jika hanya ada satu elemen, maka list menjadi kosong setelah penghapusan.
- **Delete After dan Delete Before**: *Delete After* menghapus elemen yang berada setelah elemen tertentu, sedangkan *Delete Before* menghapus elemen yang berada sebelum elemen tertentu.
- **Update, View, dan Searching**: Pada Double Linked List, operasi pembaruan (update), tampilan (view), dan pencarian (searching) dilakukan serupa dengan yang ada pada Single Linked List, tetapi memiliki kemudahan akses elemen secara

dua arah. Operasi ini meliputi:

1. **Update**: Mengubah data suatu elemen dalam list berdasarkan posisi atau nilai elemen tersebut.
2. **View**: Menampilkan seluruh elemen dalam list. Dengan akses dua arah, tampilan elemen dapat diurutkan dari awal ke akhir atau sebaliknya.
3. **Searching**: Mencari elemen dengan nilai tertentu. Proses pencarian lebih efisien karena akses data bisa dilakukan dari *first* maupun *last*, tergantung posisi elemen yang dicari.

Double Linked List memberikan kemudahan dalam proses manipulasi data dan efisiensi dalam pengaksesan, sehingga sangat cocok untuk aplikasi yang memerlukan penyisipan, penghapusan, atau pencarian data yang dinamis.

C. Guided

a. Searching

Code :

```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;
        cout << "Data " << data << " berhasil ditambahkan.\n";
    }

    // Fungsi untuk menghapus elemen dari depan list
    void deleteNode() {
        if (head == nullptr) {
            cout << "List kosong. Tidak ada data untuk dihapus.\n";
            return; // Jika list kosong
        }
        Node* temp = head;
        cout << "Data " << temp->data << " berhasil dihapus.\n";
        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;
                cout << "Data " << oldData << " berhasil diupdate menjadi " <<
newData << ".\n";
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        cout << "Data " << oldData << " tidak ditemukan.\n";
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        if (head == nullptr) {
            cout << "List sudah kosong.\n";
            return;
        }
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
        cout << "Semua data berhasil dihapus.\n";
    }

    // Fungsi untuk menampilkan semua elemen di list
    void display() {
        if (head == nullptr) {
            cout << "List kosong.\n";
            return;
        }
        Node* current = head;
        cout << "Data dalam list: ";
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "\n====== Menu ======\n";
        cout << "1. Tambah Data\n";
        cout << "2. Hapus Data\n";
        cout << "3. Update Data\n";
        cout << "4. Hapus Semua Data\n";
        cout << "5. Tampilkan Data\n";
        cout << "6. Keluar\n";
        cout << "Pilih Menu: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Masukkan data yang akan ditambahkan: ";
                cin >> data;
                list.insert(data);
                break;
            }
            case 2: {
                list.deleteNode();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Masukkan data yang akan diupdate: ";
                cin >> oldData;
                cout << "Masukkan data baru: ";
                cin >> newData;
                list.update(oldData, newData);
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                cout << "Keluar dari program.\n";
                return 0;
            }
            default: {
                cout << "Pilihan tidak valid.\n";
    return 0;
}
```

Output :

```
===== Menu =====
1. Tambah Data
2. Hapus Data
3. Update Data
4. Hapus Semua Data
5. Tampilkan Data
6. Keluar
Pilih Menu: 1
Masukkan data yang akan ditambahkan: 5
Data 5 berhasil ditambahkan.

===== Menu =====
1. Tambah Data
2. Hapus Data
3. Update Data
4. Hapus Semua Data
5. Tampilkan Data
6. Keluar
Pilih Menu: 5
Data dalam list: 5
```

Penjelasan :

Kode ini mengimplementasikan struktur Double Linked List dalam C++ dengan beberapa hal atau operasi penting yang ada didalamnya. Pertama, fungsi Insert memungkinkan penambahan elemen baru di depan list, di mana elemen pertama diatur sebagai head, dan jika list kosong, tail juga mengarah ke elemen baru ini. Kedua, fungsi DeleteNode digunakan untuk menghapus elemen pertama; apabila elemen yang dihapus adalah satu-satunya, tail diatur ke nullptr. Selain itu, terdapat fungsi Update yang mencari elemen berdasarkan nilai tertentu dan menggantinya dengan nilai baru jika ditemukan.

Selanjutnya, fungsi DeleteAll akan menghapus semua elemen dalam list, mengosongkan head dan tail setelah semua elemen dihapus. Fungsi Display menampilkan seluruh elemen dalam list, atau menampilkan pesan jika list kosong. Di bagian main atau utama dari kode ini berisi menu dengan pilihan untuk menambah, menghapus, memperbarui, menampilkan, dan menghapus semua data, serta keluar dari program. Dari kode ini mencakup operasi dasar Double Linked List yang mendukung penambahan, penghapusan, pembaruan, dan penampilan data secara dinamis.

D. Unguided

Pada latihan ini jawaban nomor 1 – 3 masih ada keterhubungan jadi untuk jawaban akan menjadi 1 file tetapi akan dijelaskan masing masing hal yang diminta, yaitu :

1. **Membuat Double Linked List**

Code:

Doublelist.cpp :

```
#include "doublelist.h"

void CreateList(list *L) {
    first(*L) = Nil;
    last(*L) = Nil;
}

address Alokasi(infotype X) {
    address P = (address) malloc(sizeof(struct
elmlist));
    if (P != Nil) {
        info(P) = X;
        next(P) = Nil;
        prev(P) = Nil;
    }
    return P;
}

void Dealokasi(address P) {
    free(P);
}

void InsertLast(list *L, address P) {
    if (first(*L) == Nil) {
        first(*L) = P;
        last(*L) = P;
    } else {
        next(last(*L)) = P;
        prev(P) = last(*L);
        last(*L) = P;
    }
}

void PrintInfo(list L) {
    address P = first(L);
    while (P != Nil) {
        printf("Nomor Polisi: %s\n", info(P).nopol);
        printf("Warna Kendaraan: %s\n", info(P).warna);
        printf("Tahun Kendaraan: %d\n",
info(P).thnBuat);
        printf("\n");
        P = next(P);
    }
}
```

Telkom University Purwokerto

Doublelist.h :

```h
#ifndef doublelist_H
#define doublelist_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define prev(P) (P)->prev
#define first(L) ((L).first)
#define last(L) ((L).last)

typedef struct {
    char nopol[10];
    char warna[10];
    int thnBuat;
} 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 InsertLast(list *L, address P);
void PrintInfo(list L);
address findElm(list L, char *nopol);

#endif
```

Main.cpp :

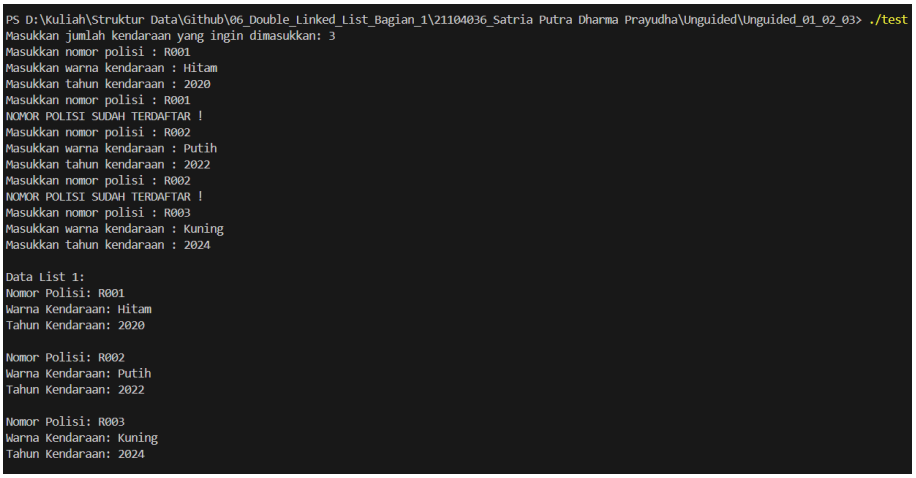
Telkom University Purwokerto

```cpp
#include <iostream>
#include "doublelist.h"

using namespace std;

int main() {
    list L;
    CreateList(&L);

    int jumlahKendaraan;
    cout << "Masukkan jumlah kendaraan yang ingin dimasukkan: ";
    cin >> jumlahKendaraan;

    char nopol[10];
    char warna[10];
    int thnBuat;
    infotype kendaraan;
    address P;

    for (int i = 0; i < jumlahKendaraan; i++) {
        cout << "Masukkan nomor polisi : ";
        cin >> nopol;

        // Mengecek apakah nomor polisi sudah ada
        if (findElm(L, nopol) != Nil) {
            cout << "NOMOR POLISI SUDAH TERDAFTAR !" << endl;
            i--;  // Meminta input ulang jika nomor polisi sudah terdaftar
            continue;
        }

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

        // Mengisi data kendaraan
        strcpy(kendaraan.nopol, nopol);
        strcpy(kendaraan.warna, warna);
        kendaraan.thnBuat = thnBuat;

        // Menambahkan data ke dalam list
        P = Alokasi(kendaraan);
        InsertLast(&L, P);
    }

    return 0;
}
```

Main.cpp :

```cpp
#include <iostream>
#include "doublelist.h"

using namespace std;

int main() {
    list L;
    CreateList(&L);

    int jumlahKendaraan;
    cout << "Masukkan jumlah kendaraan yang ingin dimasukkan: ";
    cin >> jumlahKendaraan;

    char nopol[10];
    char warna[10];
    int thnBuat;
    infotype kendaraan;
    address P;

    for (int i = 0; i < jumlahKendaraan; i++) {
        cout << "Masukkan nomor polisi : ";
        cin >> nopol;

        // Mengecek apakah nomor polisi sudah ada
        if (findElm(L, nopol) != Nil) {
            cout << "NOMOR POLISI SUDAH TERDAFTAR !" << endl;
            i--;  // Meminta input ulang jika nomor polisi sudah terdaftar
            continue;
        }

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

        // Mengisi data kendaraan
        strcpy(kendaraan.nopol, nopol);
        strcpy(kendaraan.warna, warna);
        kendaraan.thnBuat = thnBuat;

        // Menambahkan data ke dalam list
        P = Alokasi(kendaraan);
        InsertLast(&L, P);
    }

    return 0;
}
```

Output:



Penjelasan:

1. **Inisialisasi List**:
   - Menggunakan fungsi CreateList, list diinisialisasi menjadi kosong dengan mengatur pointer first dan last ke Nil.
2. **Validasi Input Nomor Polisi**:
   - Menggunakan fungsi findElm untuk mencari nomor polisi yang diinput. Jika sudah terdaftar, pengguna diminta untuk memasukkan nomor polisi yang lain.
3. **Alokasi Memori**:
   - Fungsi Alokasi digunakan untuk mengalokasikan memori untuk node baru, yang menyimpan informasi kendaraan seperti nomor polisi, warna, dan tahun pembuatan.
4. **Penambahan Elemen ke List**:
   - Elemen baru ditambahkan ke akhir list menggunakan fungsi InsertLast. Jika list kosong, elemen pertama dan terakhir diatur menjadi elemen baru.
5. **Menampilkan Data List**:
   - Semua data dalam list ditampilkan menggunakan PrintInfo, yang mencetak atribut nomor polisi, warna kendaraan, dan tahun

pembuatan dari setiap node dalam list.

2. **Mencari Data Berdasarkan Nomor Polisi**

Code:

Ditambahkan beberapa kode ke dalam file :

Doublelist.cpp :

```
address findElm(list L, char *nopol) {
    address P = first(L);
    while (P != Nil) {
        if (strcmp(info(P).nopol, nopol) == 0) {
            return P;  // Mengembalikan alamat elemen jika ditemukan
        }
        P = next(P);
    }
    return Nil;  // Jika tidak ditemukan
}
```

Main.cpp :

```
    // Mencari elemen berdasarkan nomor polisi
    char cariNopol[10];
    cout << "\nMasukkan nomor polisi yang ingin dicari: ";
    cin >> cariNopol;

    address ditemukan = findElm(L, cariNopol);
    if (ditemukan != Nil) {
        cout << "Data ditemukan:\n";
        cout << "Nomor Polisi: " << info(ditemukan).nopol << endl;
        cout << "Warna: " << info(ditemukan).warna << endl;
        cout << "Tahun: " << info(ditemukan).thnBuat << endl;
    } else {
        cout << "Nomor polisi " << cariNopol << " tidak ditemukan!" << endl;
    }
```

Output:

```
Masukkan nomor polisi yang ingin dicari: R002
Data ditemukan:
Nomor Polisi: R002
Warna: Putih
Tahun: 2022
```

Penjelasan:

- **Fungsi findElm**:

Fungsi ini digunakan untuk mencari elemen berdasarkan nomor polisi. Program meminta input nomor polisi dari pengguna dan mencari data

kendaraan yang sesuai. Jika ditemukan, data tersebut dicetak ke layar. Jika tidak, pesan bahwa data tidak ditemukan akan ditampilkan.

3. Menghapus Elemen dari List

Code:

Ditambahkan beberapa kode ke dalam file :

Doublelist.cpp :

```
void deleteFirst(list *L, address *P) {
    *P = first(*L);
    if (first(*L) != Nil) {
        if (first(*L) == last(*L)) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            first(*L) = next(first(*L));
            prev(first(*L)) = Nil;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
}

void deleteLast(list *L, address *P) {
    *P = last(*L);
    if (first(*L) != Nil) {
        if (first(*L) == last(*L)) {
            first(*L) = Nil;
            last(*L) = Nil;
        } else {
            last(*L) = prev(last(*L));
            next(last(*L)) = Nil;
        }
        prev(*P) = Nil;
        next(*P) = Nil;
    }
}

void deleteAfter(address Prec, address *P) {
    if (next(Prec) != Nil) {
        *P = next(Prec);
        next(Prec) = next(next(Prec));
        if (next(Prec) != Nil) {
            prev(next(Prec)) = Prec;
        }
        next(*P) = Nil;
        prev(*P) = Nil;
    }
```

Doublelist.h :

```
void deleteFirst(list *L, address *P);
void deleteLast(list *L, address *P);
void deleteAfter(address Prec, address *P);
```

Main.cpp :

```
    // Menghapus elemen dengan nomor polisi tertentu
    char hapusNopol[10];
    cout << "\nMasukkan nomor polisi yang ingin dihapus: ";
    cin >> hapusNopol;

    P = findElm(L, hapusNopol);
    if (P != Nil) {
        if (P == first(L)) {
            deleteFirst(&L, &P);
        } else if (P == last(L)) {
            deleteLast(&L, &P);
        } else {
            address Prec = prev(P);  // Mencari elemen sebelum P
            deleteAfter(Prec, &P);
        }
        cout << "Data dengan nomor polisi " << hapusNopol << " berhasil dihapus.\n";
    } else {
        cout << "Nomor polisi " << hapusNopol << " tidak ditemukan!\n";
    }

    // Menampilkan kembali data list setelah penghapusan
    cout << "\nData List Setelah Penghapusan:" << endl;
    PrintInfo(L);

    return 0;
}
```

Output:

```
Masukkan nomor polisi yang ingin dihapus: R003
Data dengan nomor polisi R003 berhasil dihapus.

Data List Setelah Penghapusan:
Nomor Polisi: R001
Warna Kendaraan: Hitam
Tahun Kendaraan: 2020

Nomor Polisi: R002
Warna Kendaraan: Putih
Tahun Kendaraan: 2022
```

Penjelasan:

Fitur penghapusan elemen dalam Double Linked List ditambahkan berdasarkan nomor polisi yang diinput oleh pengguna. Proses penghapusan menggunakan tiga fungsi tergantung pada posisi elemen dalam list:

- deleteFirst:

- Menghapus elemen pertama. Jika hanya ada satu elemen, list menjadi kosong (first dan last diatur ke Nil). Jika ada lebih dari satu elemen, first diatur ke elemen kedua, dan pointer prev dari elemen baru pertama diset ke Nil.

- deleteLast

- Menghapus elemen terakhir. Jika hanya ada satu elemen, list dikosongkan. Jika ada lebih dari satu elemen, last diatur ke elemen kedua terakhir, dan pointer next dari elemen baru terakhir diset ke Nil.

- deleteAfter:

- Menghapus elemen di tengah. Elemen sebelum yang dihapus (Prec) diatur untuk menunjuk ke elemen setelah yang dihapus, memutus hubungan dengan elemen di tengah.

Setelah penghapusan, list ditampilkan kembali menggunakan PrintInfo untuk memastikan elemen yang dimaksud telah dihapus.

E. Kesimpulan

Kesimpulan dari praktikum ini adalah bahwa implementasi Double Linked List memberikan fleksibilitas dan efisiensi dalam manipulasi data, terutama untuk operasi yang membutuhkan penelusuran atau modifikasi elemen dari kedua arah, baik maju maupun mundur. Dengan menggunakan pointer prev dan next pada setiap elemen, Double Linked List memungkinkan akses langsung ke elemen sebelumnya atau berikutnya, membuatnya lebih baik dalam situasi yang memerlukan penyisipan dan penghapusan data secara dinamis di berbagai posisi list dibandingkan dengan struktur list lainnya, seperti Single Linked List.

Selain itu, program ini berhasil mengimplementasikan operasi dasar dari Double Linked List, seperti penambahan, penghapusan, pembaruan, dan penampilan data. Melalui percobaan yang sudah ada pada fungsi-fungsi tersebut, dapat disimpulkan bahwa Double Linked List sangat cocok untuk aplikasi yang membutuhkan manipulasi data dengan akses yang lebih cepat dan fleksibel, serta penanganan data dalam struktur berurutan yang memungkinkan pergerakan dua arah.