Telkom University Purwokerto

Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar

13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

LAPORAN PRAKTIKUM
MODUL 6
DOUBLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :

Izzaty Zahara Br Barus – 2311104052

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

PROGRAM STUDI SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

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

II. LANDASAN TEORI

1.1 Double Linked List

Double Linked List adalah struktur data di mana setiap elemen memiliki dua pointer, yaitu prev dan next, yang menghubungkan elemen sebelumnya dan elemen berikutnya. Pada list ini, elemen pertama (first) dan elemen terakhir (last) ditandai dengan pointer khusus.

Komponen Double Linked List

- First: Pointer yang menunjuk ke elemen pertama dalam list.
- Last: Pointer yang menunjuk ke elemen terakhir dalam list.
- Next: Pointer pada setiap elemen yang menunjuk ke elemen berikutnya.
- Prev: Pointer pada setiap elemen yang menunjuk ke elemen sebelumnya.

2.1 Operasi Dasar pada Double Linked List

1. Insert (Penambahan Elemen)
    - Insert First: Menambahkan elemen di awal list.
    - Insert Last: Menambahkan elemen di akhir list.
    - Insert After/Before: Menyisipkan elemen setelah atau sebelum elemen tertentu.
2. Delete (Penghapusan Elemen)
    - Delete First: Menghapus elemen pertama dalam list.
    - Delete Last: Menghapus elemen terakhir dalam list.
    - Delete After/Before: Menghapus elemen setelah atau sebelum elemen tertentu.
3. Update, View, dan Searching (Pembaruan, Tampilan, dan Pencarian)
    - Proses pencarian, pembaruan, dan penampilan data pada double linked

list mirip dengan single linked list. Namun, pada double linked list, akses elemen lebih fleksibel karena dapat dilakukan dalam arah maju dan mundur. Struktur double linked list ini memudahkan manipulasi data, terutama dalam skenario yang membutuhkan akses bolak-balik antar elemen.

III. GUIDE

1. Guide
   a. Syntax

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

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

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

    DoublyLinkedList()  {
        head = nullptr;
        tail = nullptr;
    }

    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;
    }

    void deleteNode() {
```

```
        if(head == nullptr) {
            return;
        }
        Node* temp = head;
        head = head->next;
        if(head != nullptr) {
            head->prev = nullptr;
        }
        else {
            tail = nullptr;
        }
        delete temp;
    }

    bool update(int oldData, int newData) {
        Node* current = head;
        while(current != nullptr) {
            if(current->data == oldData) {
                current->data = newData;
                return true;
            }
            current = current->next;
        }
        return false;
    }

    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    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; //Menampilkan Penambahan
Data
        cout << "2. Delete data" << endl; //Menampilkan
Penghapusan Data
        cout << "3. Update data" << endl; //Menampilkan Pembaruan
Data
        cout << "4. Clear data" << endl; //Menampilkan
Pembersihan Data
        cout << "5. Display data" << endl; //Menampilkan Data
Yang Sudah Di Masukkan
        cout << "6. Exit" << endl; //Menampilkan Keluar Dari
Program

        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;
}
```

b. Penjelasan

Kode di atas adalah implementasi dari Doubly Linked List (Double Linked List) dalam C++. Struktur ini menyimpan elemen yang saling terhubung ke depan dan belakang melalui pointer next dan prev. Berikut penjelasan dari setiap bagian:

1. Class Node

   Node adalah struktur dasar dari linked list, berisi:

   - data : menyimpan nilai elemen.
   - prev : pointer ke elemen sebelumnya.
   - next : pointer ke elemen berikutnya.

2. Class DoublyLinkedList

   DoublyLinkedList adalah kelas utama untuk mengelola list ini, memiliki:

   - head : pointer ke elemen pertama.
   - tail : pointer ke elemen terakhir.

3. Constructor

   Inisialisasi awal head dan tail dengan nullptr menandakan bahwa list kosong.

4. Method insert(int data)

   Menambahkan elemen baru di awal list:

   - Membuat node baru.
   - Menghubungkan next dari node baru ke head.
   - Jika list tidak kosong, prev dari head akan menunjuk ke node baru.

- Jika kosong, tail akan menjadi node baru.
- head diperbarui ke node baru.

5. Method deleteNode()

   Menghapus elemen pertama:

   - Jika head kosong, maka list sudah kosong.
   - Memperbarui head ke elemen berikutnya dan menghapus node pertama.
   - Jika head baru tidak kosong, prev diatur ke nullptr.
   - Jika kosong, tail juga diatur ke nullptr.

6. Method update(int oldData, int newData)

   Mengganti data elemen:

   - Mencari node dengan nilai oldData.
   - Jika ditemukan, nilai data diperbarui menjadi newData.
   - Jika tidak ditemukan, mengembalikan nilai false.

7. Method deleteAll()

   Menghapus semua elemen:

   - Mengiterasi semua node dan menghapusnya satu per satu.
   - Mengatur head dan tail menjadi nullptr.

8. Method display()

   Menampilkan semua data dalam list:

   - Mengiterasi dari head ke tail, mencetak nilai data setiap node.

9. Main Program

   Menyediakan antarmuka konsol untuk berinteraksi dengan linked list:

   - Pengguna bisa memilih untuk menambah, menghapus, memperbarui, membersihkan, atau menampilkan data.
   - Pilihan 6 keluar dari program.

   Program ini berfungsi sebagai menu-driven program untuk mengelola Double Linked List, memberikan opsi kepada pengguna untuk mengelola elemen dalam list sesuai kebutuhan.

c. Output

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

IV. UNGUIDED

1. Task1

A.Syntax

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>

using namespace std;

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

struct ElmList;

typedef ElmList* address;

struct ElmList {
    Kendaraan info;
    address next;
    address prev;
};

struct List {
    address First;
    address Last;
};

void createList(List& L);
address alokasi(const Kendaraan& x);
void dealokasi(address& P);
void printInfo(const List& L);
void insertLast(List& L, address P);
address findElm(const List& L, const string& nopol);
void deleteFirst(List& L, address& P);
void deleteLast(List& L, address& P);

#endif
```

Telkom University Purwokerto

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

using namespace std;

void createList(List& L) {
    L.First = nullptr;
    L.Last = nullptr;
}

address alokasi(const Kendaraan& x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

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

void printInfo(const List& L) {
    if (L.First == nullptr) {
        cout << "List Kosong" << endl;
        return;
    }

    address P = L.First;
    cout << "\nIsi List : " << endl;
    while (P != nullptr) {
        cout << "Nomor Polisi  : " << P->info.nopol << endl;
        cout << "Warna         : " << P->info.warna << endl;
        cout << "Tahun Pembuatan: " << P->info.thnBuat << endl;
        cout << "-----------------------" << endl;
        P = P->next;
    }
}

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

address findElm(const List& L, const string& nopol) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

void deleteFirst(List& L, address& P) {
    if (L.First == nullptr) {
        cout << "List Kosong" << endl;
        return;
    }

    P = L.First;
    if (L.First == L.Last) {
        L.First = nullptr;
        L.Last = nullptr;
    } else {
        L.First = L.First->next;
        L.First->prev = nullptr;
        P->next = nullptr;
    }
}

void deleteLast(List& L, address& P) {
    if (L.First == nullptr) {
        cout << "List Kosong" << endl;
        return;
    }

    P = L.Last;
    if (L.First == L.Last) {
        L.First = nullptr;
        L.Last = nullptr;
    } else {
        L.Last = L.Last->prev;
        L.Last->next = nullptr;
        P->prev = nullptr;
    }
}

void deleteAfter(List& L, address Prec, address& P) {
    if (Prec == nullptr || Prec->next == nullptr) {
        cout << "Tidak ada elemen yang bisa dihapus" << endl;
        return;
    }

    P = Prec->next;
    Prec->next = P->next;

    if (P->next != nullptr) {
        P->next->prev = Prec;
    } else {
        L.Last = Prec;  // Update Last jika menghapus elemen terakhir
    }

    P->next = nullptr;
    P->prev = nullptr;
}
```

Telkom University Purwokerto

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

int main() {
    List L;
    address P;
    int pilihan;

    createList(L);

    do {
        cout << "\n====== MENU DOUBLE LINKED LIST ======" << endl;
        cout << "1. Tambah Data Kendaraan" << endl;
        cout << "2. Tampilkan Data Kendaraan" << endl;
        cout << "3. Cari Data Kendaraan" << endl;
        cout << "4. Hapus Data Kendaraan" << endl;
        cout << "0. Keluar" << endl;
        cout << "Pilihan: ";
        cin >> pilihan;

        switch(pilihan) {
            case 1: {
                Kendaraan kendaraan;
                cout << "\n--- Tambah Data Kendaraan ---" << endl;
                cout << "Nomor Polisi    : "; cin >> kendaraan.nopol;
                cout << "Warna           : "; cin >> kendaraan.warna;
                cout << "Tahun Pembuatan : "; cin >> kendaraan.thnBuat;

                insertLast(L, alokasi(kendaraan));
                cout << "\nData berhasil ditambahkan!" << endl;
                break;
            }

            case 2: {
                cout << "\n--- Data Kendaraan ---" << endl;
                printInfo(L);
                break;
            }

            case 3: {
                string nopol;
                cout << "\n--- Cari Data Kendaraan ---" << endl;
                cout << "Masukkan Nomor Polisi: ";
                cin >> nopol;

                address hasil = findElm(L, nopol);
                if(hasil != nullptr) {
                    cout << "\nKendaraan ditemukan!" << endl;
                    cout << "Nomor Polisi    : " << hasil->info.nopol << endl;
                    cout << "Warna           : " << hasil->info.warna << endl;
                    cout << "Tahun Pembuatan : " << hasil->info.thnBuat << endl;
                } else {
                    cout << "\nKendaraan tidak ditemukan!" << endl;
                }
                break;
            }

            case 4: {
                int subPilihan;
                cout << "\n--- Hapus Data Kendaraan ---" << endl;
                cout << "1. Hapus Data Pertama" << endl;
                cout << "2. Hapus Data Terakhir" << endl;
                cout << "Pilihan: ";
                cin >> subPilihan;

                if(subPilihan == 1) {
                    deleteFirst(L, P);
                    if(P != nullptr) {
                        cout << "\nData pertama berhasil dihapus!" << endl;
                        dealokasi(P);
                    }
                } else if(subPilihan == 2) {
                    deleteLast(L, P);
                    if(P != nullptr) {
                        cout << "\nData terakhir berhasil dihapus!" << endl;
                        dealokasi(P);
                    }
                }
                break;
            }

            case 0: {
                cout << "\nTerima kasih!" << endl;
                break;
            }

            default: {
                cout << "\nPilihan tidak valid!" << endl;
                break;
            }
        }
    } while(pilihan != 0);

    return 0;
}
```

B.Penjelasan

1. **doublelist.h**:

- Berisi definisi struktur data untuk menyimpan informasi kendaraan dalam bentuk double linked list.
- Struktur Kendaraan memiliki atribut nomor polisi (nopol), warna (warna), dan tahun pembuatan (thnBuat).
- Struktur ElmList digunakan sebagai elemen dari linked list, yang menyimpan informasi kendaraan dan pointer ke elemen berikutnya (next) dan sebelumnya (prev).
- Struktur List menyimpan pointer ke elemen pertama (First) dan terakhir (Last) dalam linked list.
- Fungsi prototipe meliputi fungsi untuk membuat list (createList), mengalokasikan dan mendealokasi elemen, menampilkan informasi, menyisipkan dan menghapus elemen, serta mencari elemen berdasarkan nomor polisi.

2. **doublelist.cpp**:

- Implementasi dari fungsi-fungsi yang dideklarasikan di doublelist.h.
- createList menginisialisasi linked list kosong.
- alokasi mengalokasikan memori untuk elemen baru dan mengisi informasi kendaraan.
- dealokasi menghapus elemen dari memori.
- printInfo menampilkan semua data kendaraan dalam list.
- insertLast menambahkan elemen baru di akhir list.
- findElm mencari elemen berdasarkan nomor polisi.
- deleteFirst dan deleteLast menghapus elemen pertama dan terakhir dari list.
- deleteAfter menghapus elemen di antara dua elemen tertentu.

3. **main.cpp**:

- Program utama untuk menjalankan operasi pada double linked list.
- Menu yang disediakan mencakup:
  - Menambah data kendaraan.
  - Menampilkan semua data kendaraan.
  - Mencari data kendaraan berdasarkan nomor polisi.
  - Menghapus data kendaraan (dengan pilihan menghapus data pertama

atau terakhir).

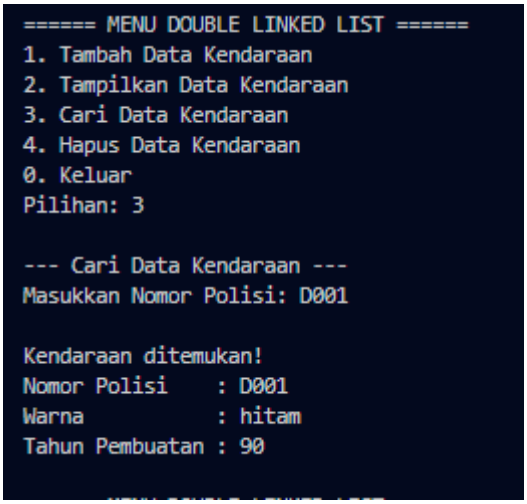
- Program berlanjut sampai pengguna memilih untuk keluar (0).

Program ini berfungsi sebagai aplikasi sederhana untuk mengelola data kendaraan dalam bentuk double linked list.

C.Output

1. Menambah Data

2. Mencari Data



V. KESIMPULAN

Pada praktikum ini, kami mempelajari konsep dasar double linked list, yaitu struktur data yang memungkinkan setiap elemen terhubung baik ke elemen sebelumnya maupun ke elemen berikutnya melalui pointer `prev` dan `next`. Hal ini membuat double linked list lebih fleksibel dibandingkan single linked list, terutama dalam hal mengakses data secara maju dan mundur. Kami juga mengimplementasikan double linked list menggunakan bahasa C++, yang melibatkan pembuatan node, penambahan node di awal dan akhir list, penghapusan node, pembaruan data dalam node, dan penelusuran list untuk menampilkan data. Operasi-operasi dasar ini menjadi penting untuk memahami cara kerja double linked list.

Selain itu, kami belajar bahwa double linked list memanfaatkan dua pointer pada setiap node untuk menunjuk ke elemen sebelumnya dan berikutnya. Penggunaan pointer ini menjadi dasar bagi double linked list untuk menghubungkan node secara dinamis, yang memungkinkan perubahan struktur data tanpa perlu menggeser seluruh

elemen seperti pada array. Fleksibilitas double linked list sangat terasa dalam manipulasi data, karena kita dapat dengan mudah menambah atau menghapus elemen di awal, akhir, atau di antara node-node tertentu, sehingga operasi seperti `insert`, `delete`, dan `update` dapat dilakukan dengan lebih efisien.

Implementasi double linked list pada studi kasus manajemen data kendaraan memberikan gambaran nyata mengenai manfaatnya dalam skenario aplikasi yang memerlukan penambahan, penghapusan, dan pencarian data secara dinamis. Selama implementasi, beberapa kesalahan yang sering terjadi adalah dereferensi pointer null dan pengaturan pointer `prev` atau `next` yang tidak tepat, yang dapat menyebabkan list rusak atau crash. Melalui percobaan ini, kami belajar untuk lebih teliti dalam pengelolaan pointer.

Secara keseluruhan, double linked list adalah struktur data yang sangat berguna untuk aplikasi yang membutuhkan fleksibilitas tinggi dalam manipulasi data. Dengan menguasai implementasi dan manipulasi double linked list, kita dapat membangun aplikasi yang lebih dinamis dan efisien dalam mengelola data yang berubah-ubah.