LAPORAN PRAKTIKUM

PERTEMUAN 6

Disusun Oleh:

Naura Aisha Zahira (2311104078)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. Tujuan

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

2. Landasan Teori

Linked list adalah jenis struktur data yang terdiri dari rangkaian elemen yang dikenal sebagai *node*, di mana setiap *node* menyimpan data dan memiliki satu atau lebih pointer yang menunjuk ke *node* lain dalam daftar. Pada *double linked list* atau daftar berantai ganda, setiap *node* dilengkapi dengan dua pointer: satu mengarah ke *node* sebelumnya, dan satu lagi mengarah ke *node* berikutnya. Struktur ini memungkinkan pergerakan dua arah, baik dari *head* (node pertama) ke *tail* (node terakhir) maupun sebaliknya.

Kelebihan *double linked list* dibandingkan *single linked list* terletak pada fleksibilitas dalam melakukan operasi seperti menambah, menghapus, atau mencari elemen dari kedua arah. Implementasinya dalam bahasa C memerlukan pengelolaan pointer yang lebih kompleks, karena setiap perubahan pada elemen perlu memperhatikan kedua pointer tersebut.

Dalam penerapan *double linked list* di bahasa C, biasanya digunakan struktur data atau kelas dengan variabel yang menyimpan data serta pointer untuk *node* sebelumnya dan berikutnya. Beberapa operasi dasar yang dapat dilakukan pada *double linked list* meliputi penambahan elemen di awal atau akhir daftar, penghapusan elemen, pembaruan data pada elemen tertentu, serta menampilkan seluruh isi daftar.

3. Guided

Code ini adalah implementasi dari Doubly Linked List (daftar berantai ganda) dalam bahasa C++. Kelas ini mendukung beberapa operasi umum pada linked list, seperti menambah, menghapus, mengupdate, dan menampilkan elemen.

Code:

```cpp
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;
}
```

```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;
    }
};
```

Penjelasan Code:

1) Class Node:
    - Node adalah kelas untuk elemen (node) dalam linked list.
    - Setiap Node memiliki:
        a. data: untuk menyimpan data.
        b. prev: pointer ke elemen sebelumnya.
        c. next: pointer ke elemen berikutnya.
2) Class DoublyLinkedList:
    - DoublyLinkedList adalah kelas utama untuk list ganda.
    - Memiliki dua pointer:
        a. head: menunjuk ke elemen pertama di list.
        b. tail: menunjuk ke elemen terakhir di list.
3) Constructor:
    - Konstruktor DoublyLinkedList() menginisialisasi head dan tail menjadi nullptr (list kosong).
4) Fungsi-fungsi dalam DoublyLinkedList:
    - insert(int data): Menambahkan elemen baru di depan (head) list.
        a. Membuat node baru dengan data yang diinput pengguna.
        b. Menghubungkan node baru ke head (elemen pertama), dan jika head kosong, tail juga menunjuk ke node baru.
    - deleteNode(): Menghapus elemen pertama dari list.
        a. Jika list kosong, fungsi langsung selesai.
        b. Menghapus head dan memindahkan head ke elemen berikutnya. Jika elemen yang dihapus adalah satu-satunya elemen, tail juga diatur menjadi nullptr.
    - update(int oldData, int newData): Mengupdate data dalam list.
        a. Mencari node dengan data tertentu (oldData). Jika ditemukan, nilai data diubah menjadi newData.
        b. Mengembalikan true jika data ditemukan dan diupdate; false jika data tidak ditemukan.
    - deleteAll(): Menghapus semua elemen di list.
        a. Mengiterasi semua node dan menghapusnya satu per satu, kemudian mengatur head dan tail menjadi nullptr.
    - display(): Menampilkan seluruh elemen di list.

a. Mengiterasi dari head ke tail, menampilkan nilai data setiap node.

5) Fungsi main():

- Menyediakan antarmuka interaktif untuk pengguna agar dapat memilih operasi yang diinginkan.
- Pengguna dapat:
  a. Menambahkan data baru.
  b. Menghapus elemen pertama.
  c. Mengupdate data tertentu.
  d. Menghapus semua data di list.
  e. Menampilkan data.
  f. Keluar dari program.
- Berdasarkan pilihan, fungsi yang sesuai dalam DoublyLinkedList dipanggil.

Cara Kerja Program:

1) Program akan meminta pengguna memasukkan pilihan operasi.
2) Berdasarkan input pengguna:
   a. insert: untuk menambah elemen baru.
   b. deleteNode: untuk menghapus elemen pertama.
   c. update: untuk mengubah data pada node tertentu.
   d. deleteAll: untuk mengosongkan seluruh list.
   e. display: untuk melihat seluruh elemen dalam list.
   f. exit: untuk keluar dari program.

Output Program:

4. Unguided

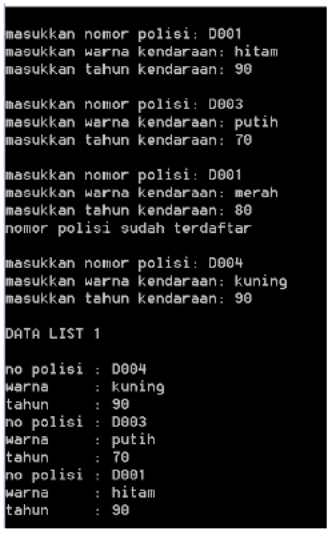
1) Buatlah ADT *Double Linked list* sebagai berikut di dalam *file* “doublelist.h”:

```
Type infotype : kendaraan <
    nopol : string
    warna : string
    thnBuat : integer
>
Type address : pointer to ElmList
Type ElmList <
    info : infotype
    next :address
    prev : address
>

Type List <
    First : address
    Last : address
>
prosedur CreateList( in/out L : List )
fungsi alokasi( x : infotype ) : address
prosedur dealokasi( in/out P : address )
prosedur printInfo( in L : List )
prosedur insertLast( in/out L : List, in P : address )
```

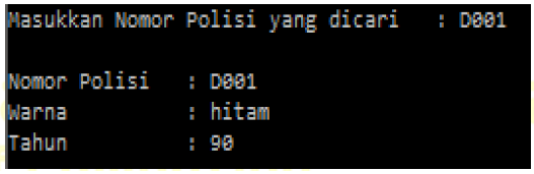
Buatlah implementasi ADT Double Linked list pada file “doublelist.cpp” dan coba hasil implementasi ADT pada file “main.cpp”.

Contoh Output:



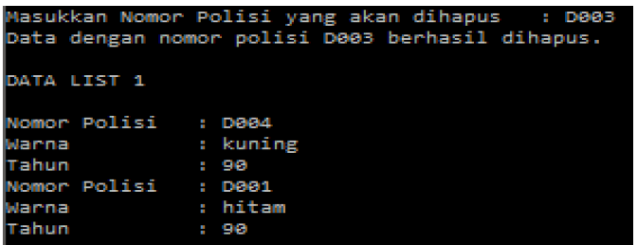
2) Carilah elemen dengan nomor polisi D001 dengan membuat fungsi baru.

fungsi findElm(L : List, x : infotype) : address



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:

1) doublelist.h

```h
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <iostream>
#include <string>

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

struct ElmList {
    infotype info;
    ElmList* next;
    ElmList* prev;
};

typedef ElmList* address;

struct List {
    address First;
    address Last;
};

void CreateList(List& L);
address alokasi(infotype x);
void dealokasi(address& P);
void printInfo(const List& L);
void insertLast(List& L, address P);
address findElm(List L, std::string nopol);
void deleteFirst(List& L, address& P);
void deleteLast(List& L, address& P);
void deleteAfter(List& L, address Prec, address& P);

#endif // DOUBLELIST_H

```

2) doublelist.cpp

```cpp
#include "doublelist.h"

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

address alokasi(infotype 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) {
    address P = L.First;
    while (P != nullptr) {
        std::cout << "no polisi: " << P->info.nopol << std::endl;
        std::cout << "warna: " << P->info.warna << std::endl;
        std::cout << "tahun: " << P->info.thnBuat << std::endl;
        P = P->next;
    }
}

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

address findElm(List L, std::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) {
        P = L.First;
        if (L.First == L.Last) {
            L.First = L.Last = nullptr;
        } else {
            L.First = L.First->next;
            L.First->prev = nullptr;
        }
        P->next = nullptr;
    }
}

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

```

3) main.cpp

```cpp
#include "doublelist.h"

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

    infotype kendaraan1 = {"D001", "hitam", 90};
    infotype kendaraan2 = {"D003", "putih", 70};
    infotype kendaraan3 = {"D004", "kuning", 90};

    insertLast(L, alokasi(kendaraan1));
    insertLast(L, alokasi(kendaraan2));
    insertLast(L, alokasi(kendaraan3));

    std::cout << "DATA LIST 1" << std::endl;
    printInfo(L);

    std::string cariNopol = "D001";
    address found = findElm(L, cariNopol);
    if (found) {
        std::cout << "Nomor Polisi yang dicari: " << cariNopol << std::endl;
        std::cout << "Warna: " << found->info.warna << std::endl;
        std::cout << "Tahun: " << found->info.thnBuat << std::endl;
    } else {
        std::cout << "Nomor Polisi tidak ditemukan." << std::endl;
    }

    address P;
    deleteFirst(L, P);
    if (P) {
        std::cout << "Data dengan nomor polisi " << P->info.nopol << " berhasil dihapus." << std::endl;
        dealokasi(P);
    }

    std::cout << "DATA LIST setelah penghapusan" << std::endl;
    printInfo(L);

    return 0;
}

```

Output Program:

```
DATA LIST 1
no polisi: D001
warna: hitam
tahun: 90
no polisi: D003
warna: putih
tahun: 70
no polisi: D004
warna: kuning
tahun: 90
Nomor Polisi yang dicari: D001
Warna: hitam
Tahun: 90
Data dengan nomor polisi D001 berhasil dihapus.
DATA LIST setelah penghapusan
no polisi: D003
warna: putih
tahun: 70
no polisi: D004
warna: kuning
tahun: 90
```

5. Kesimpulan

Praktikum ini berhasil menerapkan konsep *double linked list* dalam bahasa C. Struktur ini mempermudah navigasi dua arah dan memungkinkan pengelolaan elemen data yang lebih fleksibel. Dengan implementasi ini, mahasiswa dapat memahami konsep dasar *linked list* dan mempraktikkan pengelolaan pointer untuk memanipulasi data pada daftar berantai ganda. Selain itu, mahasiswa dapat mengaplikasikan berbagai operasi dasar pada *double linked list*, seperti menambah, menghapus, memperbarui, dan menampilkan elemen, yang penting dalam pemrograman untuk mengelola data secara dinamis.