LAPORAN PRAKTIKUM STRUKTUR DATA

PERTEMUAN 6

DOUBLE LINKED LIST (BAGIAN PERTAMA)

Nama :

Reyner Atira Prasetyo (2311104057)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

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

II. TOOL

1. Visual Studio Code
2. GCC

III. DASAR TEORI

1. Double Linked List

   Double Linked list adalah linked list yang masing – masing elemen nya memiliki 2 successor, yaitu successor yang menunjuk pada elemen sebelumnya (prev) dan successor yang menunjuk pada elemen sesudahnya (next).

2. Insert:

- Insert di awal: Node baru dibuat dan dihubungkan ke node pertama. Pointer prev dari node pertama lama diarahkan ke node baru, sementara pointer next node baru menunjuk ke node pertama lama.
- Insert di akhir: Node baru dihubungkan ke node terakhir, di mana pointer next dari node terakhir lama diarahkan ke node baru, dan pointer prev node baru menunjuk ke node terakhir lama.
- Insert di posisi tertentu: Node baru disisipkan di antara dua node yang ada, dengan pointer prev dan next dari node baru disesuaikan agar terhubung dengan benar ke node sebelum dan sesudahnya.

3. Delete:

- Delete di awal: Node pertama dihapus dengan mengarahkan pointer prev node kedua ke nullptr, dan pointer head diperbarui ke node kedua.
- Delete di akhir: Node terakhir dihapus dengan mengarahkan pointer next node sebelum node terakhir ke nullptr, dan pointer tail diperbarui.
- Delete di posisi tertentu: Node yang dihapus memiliki pointer prev node berikutnya diarahkan ke node sebelumnya, dan pointer next node sebelumnya diarahkan ke node setelah node yang dihapus.

4. Update, View, dan Searching

   Proses pencarian, update data dan view data pada dasarnya sama dengan proses pada single linked list. Hanya saja pada double linked list lebih mudah dalam melakukan proses akses elemen, karena bisa melakukan iterasi maju dan mundur.

IV. GUIDED

1. guidedM6.cpp

```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;
}
```

Output add data :

```
Data\pertemuan6>  & 'c:\Users\ASU
t-MIEngine-In-yrs43t4l.kuu' '--st
naqsb5.ypc' '--dbgExe=C:\msys64\u
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: 1
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: 3
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: 5
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

Output display data :

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

Output update data :

```
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: 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
5 3 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

Output delete data dan clear data :

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
3 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

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

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

V. UNGUIDED

1. doublelist.h

```
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#include <string>

using namespace std;

// Type infotype
```

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

// Type address
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 pada ADT Double Linked List
void CreateList(List &L);
address alokasi(kendaraan x);
void dealokasi(address &P);
bool isDuplicate(const List &L, const kendaraan &x);
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);
void deleteAfter(List &L, ElmList* prec, address &P);

#endif // DOUBLELIST_H
```

2. doublelist.cpp

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

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

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

```
    return P;
}

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

// Fungsi isDuplicate
bool isDuplicate(const List &L, const kendaraan &x) {
    address P = L.First;
    while (P != nullptr) {
        if (P->info.nopol == x.nopol) {
            return true; // Ditemukan duplikat berdasarkan nopol
        }
        P = P->next;
    }
    return false;
}

// Prosedur printInfo
void printInfo(const List &L) {
    address P = L.First;
    while (P != nullptr) {
        cout << "Nopol: " << P->info.nopol
             << "\nWarna: " << P->info.warna
             << "\nTahun: " << P->info.thnBuat << "\n" << endl;
        P = P->next;
    }
}

// Prosedur insertLast
void insertLast(List &L, address P) {
    if (P == nullptr) return;

    if (isDuplicate(L, P->info)){
        cout << "Nopol: " << P->info.nopol << " sudah terdaftar. Data
tidak ditambahkan.\n" << endl;
        dealokasi(P);
        return;
    }

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

// Fungsi untuk mencari elemen dalam list
address findElm(const List &L, const string &nopol) {
    address P = L.First;
```

```
    while (P != nullptr) {
        if (P->info.nopol == nopol) {
            return P; // Ditemukan
        }
        P = P->next;
    }
    return nullptr; // Tidak ditemukan
}

// Prosedur untuk menghapus elemen pertama
void deleteFirst(List &L, address &P) {
    if (L.First == nullptr) {
        cout << "List kosong, tidak ada yang bisa dihapus." << endl;
        return;
    }
    P = L.First; // Simpan elemen pertama
    L.First = L.First->next;
    if (L.First != nullptr) {
        L.First->prev = nullptr;
    } else {
        L.Last = nullptr; // Jika list menjadi kosong
    }
    dealokasi(P); // Hapus elemen
    cout << "Elemen pertama dihapus." << endl;
}

// Prosedur untuk menghapus elemen terakhir
void deleteLast(List &L, address &P) {
    if (L.Last == nullptr) {
        cout << "List kosong, tidak ada yang bisa dihapus." << endl;
        return;
    }
    P = L.Last; // Simpan elemen terakhir
    L.Last = L.Last->prev;
    if (L.Last != nullptr) {
        L.Last->next = nullptr;
    } else {
        L.First = nullptr; // Jika list menjadi kosong
    }
    dealokasi(P); // Hapus elemen
    cout << "Elemen terakhir dihapus." << endl;
}

// Prosedur untuk menghapus elemen setelah elemen tertentu
void deleteAfter(List &L, ElmList* prec, address &P) {
    if (prec == nullptr || prec->next == nullptr) {
        cout << "Tidak ada elemen setelah elemen yang diberikan." <<
endl;
        return;
    }
    P = prec->next; // Simpan elemen setelah
    prec->next = P->next;
    if (P->next != nullptr) {
        P->next->prev = prec;
    } else {
        L.Last = prec; // Jika dihapus adalah elemen terakhir
```

```
    }
    dealokasi(P); // Hapus elemen
    cout << "Elemen setelah elemen yang diberikan dihapus." << endl;
}
```

3. main.cpp

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

using namespace std;

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

    int numEntries;
    cout << "Masukkan jumlah kendaraan yang ingin ditambahkan: ";
    cin >> numEntries;
    cin.ignore(); // Mengabaikan karakter newline setelah input integer

    for (int i = 0; i < numEntries; i++) {
        kendaraan k;
        cout << "\nMasukkan data kendaraan ke-" << (i + 1) << ":\n";

        cout << "Nomor Polisi: ";
        getline(cin, k.nopol);

        cout << "Warna: ";
        getline(cin, k.warna);

        cout << "Tahun Pembuatan: ";
        cin >> k.thnBuat;
        cin.ignore(); // Mengabaikan karakter newline setelah input
integer

        // Mengalokasikan elemen dan memasukkan ke dalam list
        address P = alokasi(k);
        insertLast(L, P);
    }

    // Menampilkan isi list
    cout << "\nDATA LIST 1\n";
    cout << "--------------------\n";
    printInfo(L);

    string nopolToFind;
    cout << "Masukkan nopol kendaraan yang ingin dicari: ";
    getline(cin, nopolToFind);

    address foundP = findElm(L, nopolToFind); // Rename to foundP

    if (foundP != nullptr) {
        cout << "Data kendaraan ditemukan:\n";
        cout << "Nopol: " << foundP->info.nopol
```

```
        << "\nWarna: " << foundP->info.warna
        << "\nTahun: " << foundP->info.thnBuat << "\n";
    } else {
        cout << "Elemen dengan nopol " << nopolToFind << " tidak
ditemukan.\n";
    }

    // Mencari dan menghapus elemen dengan nopol yang diinput
    string nopolToDelete;
    cout << "Masukkan nopol kendaraan yang ingin dihapus: ";
    getline(cin, nopolToDelete);
    address deleteP = findElm(L, nopolToDelete); // Rename to deleteP

    if (deleteP != nullptr) {
        // Jika elemen yang ingin dihapus adalah elemen pertama
        if (deleteP == L.First) {
            deleteFirst(L, deleteP); // Ensure you pass the right
address
        }
        // Jika elemen yang ingin dihapus adalah elemen terakhir
        else if (deleteP == L.Last) {
            deleteLast(L, deleteP); // Ensure you pass the right
address
        }
        // Jika elemen yang ingin dihapus berada di tengah list
        else {
            deleteAfter(L, deleteP->prev, deleteP); // Menghapus
setelah node predecessor
        }

        cout << "Elemen dengan nopol " << nopolToDelete << " berhasil
dihapus.\n";
    } else {
        cout << "Elemen dengan nopol " << nopolToDelete << " tidak
ditemukan.\n";
    }

    // Menampilkan isi list setelah penghapusan
    cout << "\nDATA LIST SETELAH PENGHAPUSAN\n";
    cout << "--------------------\n";
    printInfo(L);

    return 0;
}
```

Output :

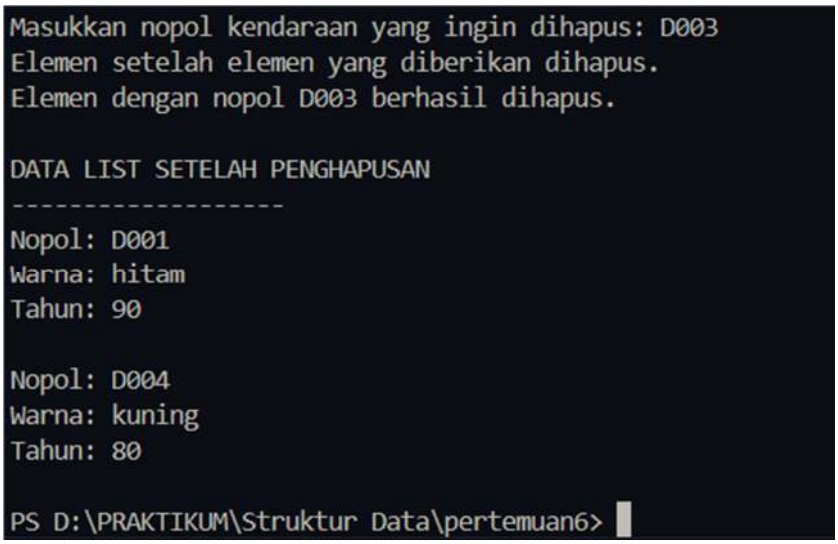
1. Implementasi

```
Masukkan jumlah kendaraan yang ingin ditambahkan: 4

Masukkan data kendaraan ke-1:
Nomor Polisi: D001
Warna: hitam
Tahun Pembuatan: 90

Masukkan data kendaraan ke-2:
Nomor Polisi: D003
Warna: putih
Tahun Pembuatan: 70

Masukkan data kendaraan ke-3:
Nomor Polisi: D001
Warna: merah
Tahun Pembuatan: 80
Nopol: D001 sudah terdaftar. Data tidak ditambahkan.


Masukkan data kendaraan ke-4:
Nomor Polisi: D004
Warna: kuning
Tahun Pembuatan: 80

DATA LIST 1
------------------
Nopol: D001
Warna: hitam
Tahun: 90

Nopol: D003
Warna: putih
Tahun: 70

Nopol: D004
Warna: kuning
Tahun: 80
```

2. Search elemen :

```
Masukkan nopol kendaraan yang ingin dicari: D001
Data kendaraan ditemukan:
Nopol: D001
Warna: hitam
Tahun: 90
```

3. Delete elemen tertentu :



VI. KESIMPULAN

Pada praktikum ini, saya telah berhasil mengimplementasikan dan mempraktikkan pembuatan double linked list, prosedur insert dan delete, serta metode find element pada double linked list menggunakan bahasa pemrograman C++. Saya mempelajari bagaimana membuat, mengalokasikan, menambahkan elemen di awal, akhir, dan tengah list, mencetak elemen, serta mencari elemen dalam list. Dari hasil implementasi, saya memahami bagaimana struktur data ini bekerja, khususnya dalam menyusun dan mengakses elemen-elemen yang dihubungkan secara berurutan. Dengan menerapkan pencarian menggunakan loop, saya dapat menemukan elemen yang dicari dalam list. Praktikum ini menunjukkan pentingnya penggunaan pointer dalam pengelolaan memori dinamis dan bagaimana teknik dasar pencarian diterapkan pada struktur data linked list.