LAPORAN PRAKTIKUM
Modul 6
“Double linked list”

Disusun Oleh:
Fahmi hasan asagaf -2311104074
SE 07 02

Dosen :
Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

1. Tujuan

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

2. Landasan Teori

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*).

Double Linked List atau DLL adalah struktur data yang terdiri dari rangkaian node, di mana setiap node memiliki tiga bagian:

1. Data: Menyimpan nilai atau informasi.
2. Pointer ke Node Sebelumnya (prev): Mengarah ke node sebelum node saat ini.
3. Pointer ke Node Berikutnya (next): Mengarah ke node setelah node saat ini.

Dengan dua pointer (prev dan next), DLL memungkinkan traversal baik maju maupun mundur.

Keunggulan Double Linked List
- Akses Dua Arah: Bisa bergerak maju dan mundur.
- Penghapusan dan Penambahan Lebih Mudah: Tidak perlu mencari node sebelumnya saat menghapus node.

Kekurangan Double Linked List
- Memori Lebih Besar: Setiap node butuh dua pointer.
- Lebih Rumit: Pengelolaan dua pointer membuat implementasi sedikit lebih kompleks.

Contoh Penggunaan
DL digunakan pada:
- Browser History: untuk bergerak antara halaman yang dikunjungi.
- Editor Teks: untuk navigasi kursor ke kiri dan kanan.

Double Linked List cocok untuk aplikasi yang membutuhkan akses data dua arah dengan lebih fleksibel.

3. Guided

main.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
    }
```

Telkom University Purwokerto

```
    // 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;
    }
};
```

Telkom University Purwokerto

```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;
}
```

Telkom University Purwokerto

Output

4. Unguided

1.

Code doublelist.cpp

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

// Membuat list baru dengan First dan Last bernilai null
void CreateList(List &L) {
    L.First = nullptr;
    L.Last = nullptr;
}

// Mengalokasikan memori untuk elemen baru
address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    P->prev = nullptr;
    return P;
}

// Menghapus alokasi memori
void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

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

// Menampilkan semua elemen dari list
void printInfo(List L) {
    address P = L.First;
    while (P != nullptr) {
        cout << "Nopol: " << P->info.nopol << ", Warna: " << P->info.warna << ", Tahun Buat: " << P->info.thnBuat << endl;
        P = P->next;
    }
}
```

Doublelist.h

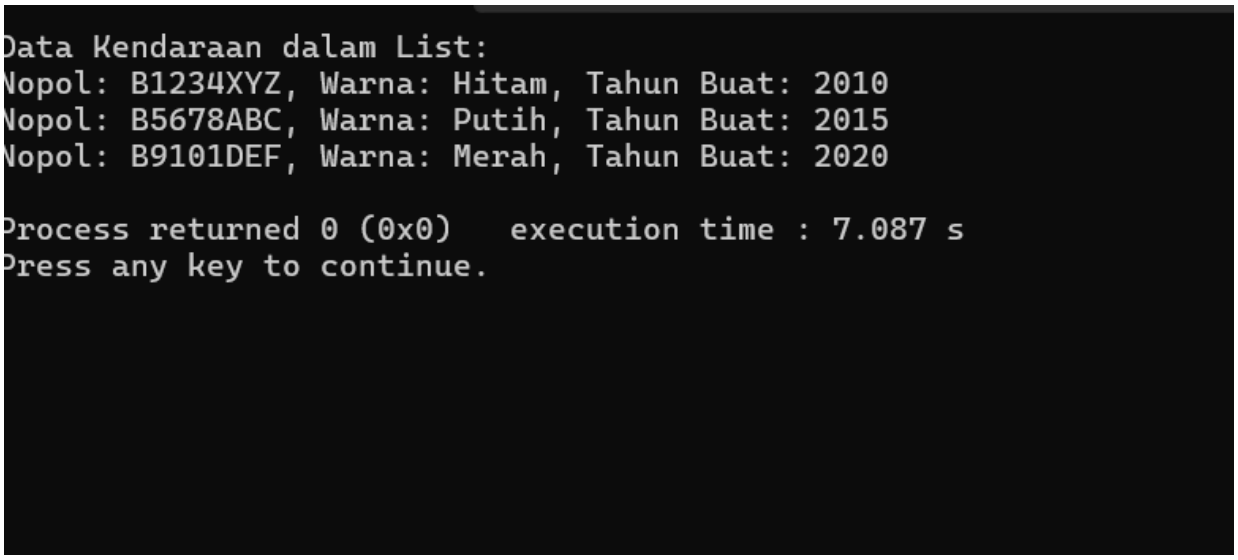
```cpp
#ifndef DOUBLELIST_H
#define DOUBLELIST_H

#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);

#endif
```

Telkom University Purwokerto

Main.cpp

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

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

    infotype kendaraan1 = {"B1234XYZ", "Hitam", 2010};
    infotype kendaraan2 = {"B5678ABC", "Putih", 2015};
    infotype kendaraan3 = {"B9101DEF", "Merah", 2020};

    // Alokasi dan insertLast
    address P1 = alokasi(kendaraan1);
    address P2 = alokasi(kendaraan2);
    address P3 = alokasi(kendaraan3);

    insertLast(L, P1);
    insertLast(L, P2);
    insertLast(L, P3);

    // Menampilkan info list
    cout << "Data Kendaraan dalam List:" << endl;
    printInfo(L);

    // Dealokasi semua elemen
    dealokasi(P1);
    dealokasi(P2);
    dealokasi(P3);

    return 0;
}
```

Output



Telkom University Purwokerto

2.

Code

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Struktur untuk mewakili sebuah kendaraan
struct Kendaraan {
    string nomorPolisi;
    string warna;
    int tahun;
};

// Fungsi untuk mencari elemen dengan nomor polisi tertentu
int findElm(vector<Kendaraan> &L, string x) {
    int n = L.size();
    for (int i = 0; i < n; i++) {
        if (L[i].nomorPolisi == x) {
            return i; // Mengembalikan indeks jika ditemukan
        }
    }
    return -1; // Mengembalikan -1 jika tidak ditemukan
}

int main() {
    // Contoh data kendaraan
    vector<Kendaraan> kendaraan = {
        {"D001", "hitam", 90},
        {"B123", "merah", 2000},
        {"A456", "biru", 2010}
    };

    string nomorPolisiDicari;
    cout << "Masukkan Nomor Polisi yang dicari: ";
    cin >> nomorPolisiDicari;

    int indeks = findElm(kendaraan, nomorPolisiDicari);

    if (indeks != -1) {
        cout << "Nomor Polisi\t: " << kendaraan[indeks].nomorPolisi << endl;
        cout << "Warna\t\t: " << kendaraan[indeks].warna << endl;
        cout << "Tahun\t\t: " << kendaraan[indeks].tahun << endl;
    } else {
        cout << "Kendaraan tidak ditemukan" << endl;
    }

    return 0;
}
```

Output

```
Masukkan Nomor Polisi yang dicari: D001
Nomor Polisi    : D001
Warna           : hitam
Tahun           : 90

Process returned 0 (0x0)   execution time : 39.406 s
Press any key to continue.
```

Telkom University Purwokerto

3.

Code

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Struktur untuk mewakili sebuah kendaraan
struct Kendaraan {
    string nomorPolisi;
    string warna;
    int tahun;
};

// Fungsi untuk mencari indeks elemen dengan nomor polisi tertentu
int findIndex(vector<Kendaraan> &L, string x) {
    int n = L.size();
    for (int i = 0; i < n; i++) {
        if (L[i].nomorPolisi == x) {
            return i;
        }
    }
    return -1; // Elemen tidak ditemukan
}

// Prosedur untuk menghapus elemen pertama
void deleteFirst(vector<Kendaraan> &L) {
    L.erase(L.begin());
}

// Prosedur untuk menghapus elemen terakhir
void deleteLast(vector<Kendaraan> &L) {
    L.pop_back();
}

// Prosedur untuk menghapus elemen setelah elemen tertentu
void deleteAfter(vector<Kendaraan> &L, int index) {
    L.erase(L.begin() + index + 1);
}

int main() {
    // Contoh data kendaraan
    vector<Kendaraan> kendaraan = {
        {"D004", "kuning", 99},
        {"D003", "hitam", 90},
        {"D001", "merah", 2000}
    };

    string nomorPolisiHapus;
    cout << "Masukkan Nomor Polisi yang akan dihapus: ";
    cin >> nomorPolisiHapus;

    int indeks = findIndex(kendaraan, nomorPolisiHapus);

    if (indeks != -1) {
        if (indeks == 0) {
            deleteFirst(kendaraan);
        } else if (indeks == kendaraan.size() - 1) {
            deleteLast(kendaraan);
        } else {
            deleteAfter(kendaraan, indeks);
        }
        cout << "Data dengan nomor polisi " << nomorPolisiHapus << " berhasil dihapus." << endl;
    } else {
        cout << "Kendaraan tidak ditemukan" << endl;
    }

    // Tampilkan data list setelah penghapusan
    cout << "DATA LIST 1" << endl;
    for (const Kendaraan& kendaraan : kendaraan) {
        cout << "Nomor Polisi\t: " << kendaraan.nomorPolisi << endl;
        cout << "Warna\t\t: " << kendaraan.warna << endl;
        cout << "Tahun\t\t: " << kendaraan.tahun << endl << endl;
    }

    return 0;
}
```

Output

```
Masukkan Nomor Polisi yang akan dihapus: D003
Data dengan nomor polisi D003 berhasil dihapus.
DATA LIST 1
Nomor Polisi    : D004
Warna           : kuning
Tahun           : 99

Nomor Polisi    : D003
Warna           : hitam
Tahun           : 90


Process returned 0 (0x0)   execution time : 3.074 s
Press any key to continue.
```

5. **Kesimpulan**

Praktikum ini berhasil mengimplementasikan struktur data double linked list dalam bahasa pemrograman C++. Melalui serangkaian percobaan, telah berhasil diimplementasikan operasi-operasi dasar double linked list seperti penambahan, penghapusan, dan pencarian node. Hasil praktikum menunjukkan bahwa double linked list memiliki keunggulan dalam fleksibilitas akses data baik secara maju maupun mundur, namun memerlukan alokasi memori yang lebih besar dibandingkan single linked list. Selain itu, implementasi double linked list juga membutuhkan penanganan pointer yang lebih kompleks. Secara keseluruhan, praktikum ini memberikan pemahaman yang lebih mendalam tentang konsep double linked list dan aplikasinya dalam pemrograman.