LAPORAN PRAKTIKUM

MODUL III
SINGLE AND DOUBLE LINKED LIST

Disusun oleh:
Maulana Ghani Rolanda
NIM: 2311102012

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

BAB I

TUJUAN PRAKTIKUM

1.Mahasiswa memahami perbedaan konsep Single dan Double Linked List

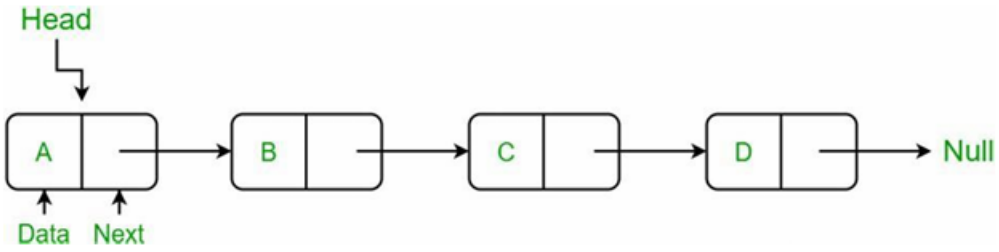
2.Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman

BAB II

DASAR TEORI

a) Single Linked List Linked List

merupakan suatu bentuk struktur data yang berisi kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung, dinamis, dan terbatas. Setiap elemen dalam linked list dihubungkan ke elemen lain melalui pointer. Masing-masing komponen sering disebut dengan simpul atau node atau verteks. Pointer adalah alamat elemen. Setiap simpul pada dasarnya dibagi atas dua bagian pertama disebut bagian isi atau informasi atau data yang berisi nilai yang disimpan oleh simpul. Bagian kedua disebut bagian pointer yang berisi alamat dari node berikutnya atau sebelumnya. Dengan menggunakan struktur seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya. Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head dan elemen terakhir dari suatu list disebut tail. Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya lebih efisien dalam penggunaan memori dibandingkan dengan jenis Linked List lainnya, seperti Double Linked List dan Circular Linked List. Single linked list yang kedua adalah circular linked list. Perbedaan circular linked list dan non circular linked adalah penunjuk next pada node terakhir pada circular linked list akan selalu merujuk ke node pertama

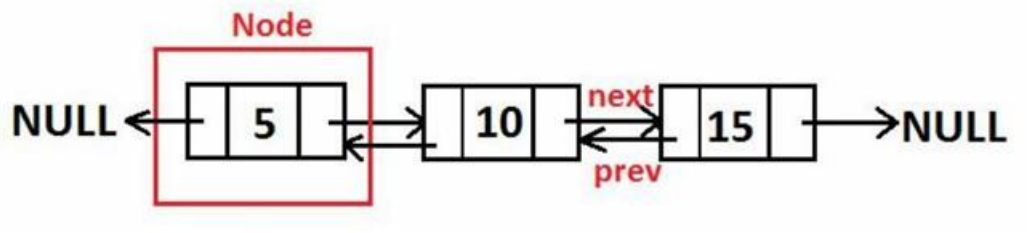


Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya lebih efisien dalam penggunaan memori dibandingkan dengan jenis Linked List lainnya, seperti Double Linked List dan Circular Linked List. Single linked list yang kedua adalah circular linked list. Perbedaan circular linked list dan non circular linked adalah penunjuk next pada node terakhir pada circular linked list akan selalu merujuk ke node pertama.

b) Double Linked List Double Linked List

adalah struktur data Linked List yang mirip dengan Single Linked List, namun dengan tambahan satu pointer tambahan pada setiap simpul yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul mana saja secara efisien. Setiap simpul pada Double Linked List memiliki tiga elemen penting, yaitu elemen data (biasanya berupa nilai), pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya. Keuntungan dari Double Linked List adalah memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul dimana saja dengan efisien, sehingga sangat

berguna dalam implementasi beberapa algoritma yang membutuhkan operasi tersebut. Selain itu, Double Linked List juga memungkinkan kita untuk melakukan traversal pada list baik dari depan (head) maupun dari belakang (tail) dengan mudah. Namun, kekurangan dari Double Linked List adalah penggunaan memori yang lebih besar dibandingkan dengan Single Linked List, karena setiap simpul membutuhkan satu pointer tambahan. Selain itu, Double Linked List juga membutuhkan waktu eksekusi yang lebih lama dalam operasi penambahan dan penghapusan jika dibandingkan dengan Single Linked List. Representasi sebuah double linked list dapat dilihat pada gambar berikut ini:



Di dalam sebuah linked list, ada 2 pointer yang menjadi penunjuk utama, yakni pointer HEAD yang menunjuk pada node pertama di dalam linked list itu sendiri dan pointer TAIL yang menunjuk pada node paling akhir di dalam linked list. Sebuah linked list dikatakan kosong apabila isi pointer head adalah NULL. Selain itu, nilai pointer prev dari HEAD selalu NULL, karena merupakan data pertama. Begitu pula dengan pointer next dari TAIL yang selalu bernilai NULL sebagai penanda data terakhir.

BAB III

GUIDED

1. Guided 1

Source code

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

/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    // komponen/member
    int data;
    string kata;
    Node *next;
};
Node *head;
Node *tail;
// Inisialisasi Node
void init()
{
    head = NULL;
    tail = NULL;
}
// Pengecekan
bool isEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}
// Tambah Depan
void insertDepan(int nilai, string kata)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
```

```
    baru->kata = kata;
 baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
    else
    {
        baru->next = head;
        head = baru;
    }
}
// Tambah Belakang
void insertBelakang(int nilai, string kata)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->kata = kata;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
    else
    {
        tail->next = baru;
        tail = baru;
    }
}
// Hitung Jumlah List
int hitungList()
{
    Node *hitung;
hitung = head;
    int jumlah = 0;
    while (hitung != NULL)
    {
        jumlah++;
```

```
        hitung = hitung->next;
    }
    return jumlah;
}
// Tambah Tengah
void insertTengah(int data, string kata, int posisi)
{
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi diluar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        Node *baru, *bantu;
        baru = new Node();
        baru->data = data;
        baru->kata = kata;
        // tranversing
        bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
}
}
// Hapus Depan
void hapusDepan()
{
    Node *hapus;
    if (isEmpty() == false)
    {
        if (head->next != NULL)
        {
```

```
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else
        {
            head = tail = NULL;
        }
    }
    else
    {
        cout << "List kosong!" << endl;
    }
}
// Hapus Belakang
void hapusBelakang()
{
    Node *hapus;
    Node *bantu;
    if (isEmpty() == false)
    {
        if (head != tail)
        {
            hapus = tail;
            bantu = head;
            while (bantu->next != tail)
            {
                bantu = bantu->next;
            }
            tail = bantu;
            tail->next = NULL;
            delete hapus;
        }
        else
        {
            head = tail = NULL;
        }
    }
    else
    {
        cout << "List kosong!" << endl;
```

```
    }
}
// Hapus Tengah
void hapusTengah(int posisi)
{
    Node *hapus, *bantu, *bantu2;
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi di luar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi)
        {
            if (nomor == posisi - 1)
 {
                bantu2 = bantu;
            }
            if (nomor == posisi)
            {
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        bantu2->next = bantu;
        delete hapus;
    }
}
// Ubah Depan
void ubahDepan(int data, string kata)
{
    if (isEmpty() == false)
    {
        head->data = data;
```

```
        head->kata = kata;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
// Ubah Tengah
void ubahTengah(int data, string kata, int posisi)
{
    Node *bantu;
    if (isEmpty() == false)
    {
        if (posisi < 1 || posisi > hitungList())
        {
  cout << "Posisi di luar jangkauan" << endl;
        }
        else if (posisi == 1)
        {
            cout << "Posisi bukan posisi tengah" << endl;
        }
        else
        {
            bantu = head;
            int nomor = 1;
            while (nomor < posisi)
            {
                bantu = bantu->next;
                nomor++;
            }
            bantu->data = data;
            bantu->kata = kata;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
// Ubah Belakang
void ubahBelakang(int data, string kata)
```

```
{
    if (isEmpty() == false)
    {
        tail->data = data;
        tail->kata = kata;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
// Hapus List
void clearList()
{
    Node *bantu, *hapus;
    bantu = head;
    while (bantu != NULL)
    {
        hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}
// Tampilkan List
void tampil()
{
    Node *bantu;
    bantu = head;
    if (isEmpty() == false)
    {
        while (bantu != NULL)
        {
            cout << bantu->data << "\t";
            cout << bantu->kata;
            bantu = bantu->next;
        }
        cout << endl;
  }
    else
```

```
    {
        cout << "List masih kosong!" << endl;
    }
}
int main()
{
    init();
    insertDepan(3, "satu");
    tampil();
    insertBelakang(5, "dua");
    tampil();
    insertDepan(2, "tiga");
    tampil();
    insertDepan(1, "empat");
    tampil();
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7, "sepuluh" , 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1, "delapan");
    tampil();
    ubahBelakang(8, "sembilan");
    tampil();
    ubahTengah(11, "tujuh", 2);
    tampil();
    return 0;
}
```

Screenshoot program

```
PS C:\olan\vscode\Modul3> cd "c:\olan\vscode\Modul3\"
3       satu
3       satu5   dua
2       tiga3   satu5   dua
1       empat2  tiga3   satu5   dua
2       tiga3   satu5   dua
2       tiga3   satu
2       tiga7   sepuluh3        satu
2       tiga3   satu
1       delapan3        satu
1       delapan8        sembilan
1       delapan11       tujuh
```

Deskripsi program

Program tersebut merupakan implementasi dari linked list tunggal non-circular dalam bahasa C++. Ini mencakup fungsi-fungsi untuk inisialisasi, penambahan, penghapusan, perubahan, dan penampilan isi dari linked list. Setiap node menyimpan sebuah integer dan sebuah string. Program mengizinkan operasi-operasi dasar seperti penambahan di depan/belakang, di tengah, penghapusan di depan/belakang/tengah, perubahan nilai, dan menampilkan isi linked list.

2. Guided 2

Source code

```cpp
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    string kata;
    Node *prev;
    Node *next;
};
class DoublyLinkedList
{
public:
```

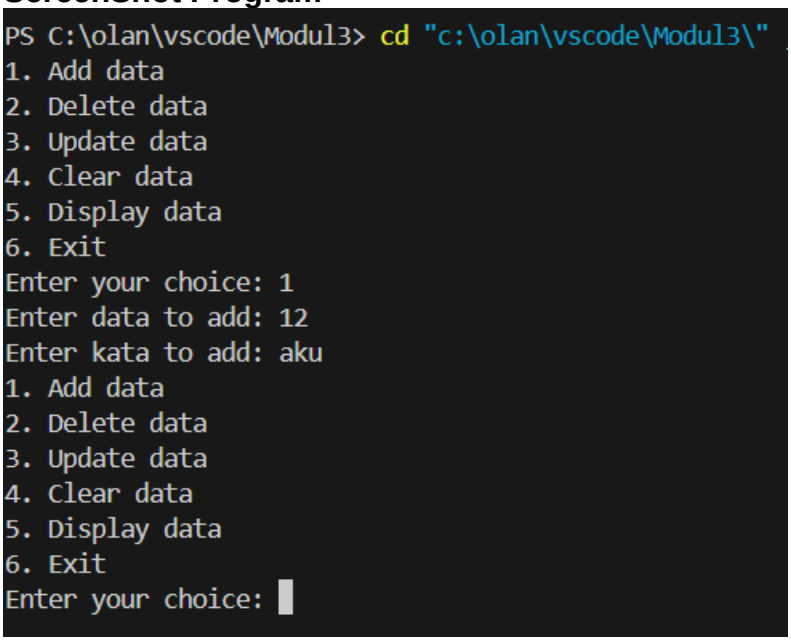
```
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void push(int data, string kata)
    {
        Node *newNode = new Node;
        newNode->data = data;
        newNode->kata = kata;
        newNode->prev = nullptr;
        newNode->next = head;
        if (head != nullptr)
        {
            head->prev = newNode;
        }
        else
        {
            tail = newNode;
        }
        head = newNode;
    }
    void pop()
    {
        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, string oldKata, string
newKata)
    {
        Node *current = head;
        while (current != nullptr)
        {
            if (current->data == oldData)
            {
                current->data = newData;
                current->kata = newKata;
                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->kata << endl;
            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;
            string kata;
            cout << "Enter data to add: ";
            cin >> data;
            cout << "Enter kata to add: ";
            cin >> kata;
            list.push(data, kata);
            break;
        }
        case 2:
        {
            list.pop();
            break;
        }
        case 3:
        {
            int oldData, newData;
            string oldKata, newKata;
            cout << "Enter old data: ";
            cin >> oldData;
            cout << "Enter new data: ";
            cin >> newData;
            cout << "Enter old Kata: ";
            cin >> oldKata;
```

```
            cout << "Enter new kata: ";
            cin >> newKata;
            bool updated = list.update (oldData, newData, oldKata,
newKata);
            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;
}
```

ScreenShot Program



Deskripsi Program

Program tersebut adalah implementasi Doubly Linked List dalam bahasa C++. Program ini memungkinkan pengguna untuk menambah, menghapus, memperbarui, menghapus semua data, dan menampilkan data dari linked list melalui menu interaktif. Setiap node dalam linked list menyimpan sebuah integer dan sebuah string. Program menggunakan konsep pointer untuk menghubungkan setiap node.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

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

// Deklarasi Struct Node
struct Node {
    string nama;
    int usia;
    Node *next;
};

// Deklarasi Linked List
class LinkedList {
private:
    Node *head;

public:
    // Konstruktor
    LinkedList() {
        head = nullptr;
    }

    // Method untuk menambah data di depan
    void insertDepan(string nama, int usia) {
        Node *baru = new Node;
        baru->nama = nama;
        baru->usia = usia;
        baru->next = head;
        head = baru;
    }

    // Method untuk menambah data di belakang
    void insertBelakang(string nama, int usia) {
        Node *baru = new Node;
```

```
        baru->nama = nama;
        baru->usia = usia;
        baru->next = nullptr;

        if (head == nullptr) {
            head = baru;
        } else {
            Node *temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = baru;
        }
    }

    // Method untuk menambah data di tengah
    void insertTengah(string nama, int usia, string posisi) {
        Node *baru = new Node;
        baru->nama = nama;
        baru->usia = usia;

        Node *temp = head;
        while (temp != nullptr && temp->nama != posisi) {
            temp = temp->next;
        }

        if (temp != nullptr) {
            baru->next = temp->next;
            temp->next = baru;
        } else {
            cout << "Data tidak ditemukan." << endl;
        }
    }

    // Method untuk menghapus data
    void hapus(string nama) {
        if (head == nullptr) {
            cout << "Linked list kosong." << endl;
            return;
        }
```

```
        if (head->nama == nama) {
            Node *hapus = head;
            head = head->next;
            delete hapus;
            return;
        }

        Node *temp = head;
        while (temp->next != nullptr && temp->next->nama != nama)
{
            temp = temp->next;
        }

        if (temp->next != nullptr) {
            Node *hapus = temp->next;
            temp->next = temp->next->next;
            delete hapus;
        } else {
            cout << "Data tidak ditemukan." << endl;
        }
    }

    // Method untuk mengubah data
    void ubahData(string nama, int usia, string namaBaru, int
usiaBaru) {
        Node *temp = head;
        while (temp != nullptr) {
            if (temp->nama == nama && temp->usia == usia) {
                temp->nama = namaBaru;
                temp->usia = usiaBaru;
                return;
            }
            temp = temp->next;
        }
        cout << "Data tidak ditemukan." << endl;
    }

    // Method untuk menampilkan seluruh data
    void tampilData() {
        Node *temp = head;
        while (temp != nullptr) {
```

```
            cout << temp->nama << "\t" << temp->usia << endl;
            temp = temp->next;
        }
    }
};

int main() {
    LinkedList list;

    // Data pertama (nama Anda dan usia Anda)
    string namaAnda;
    int usiaAnda;
    cout << "Masukkan nama Anda: ";
    getline(cin, namaAnda);
    cout << "Masukkan usia Anda: ";
    cin >> usiaAnda;
    cin.ignore(); // Membersihkan buffer input sebelumnya

    // Masukkan data pertama ke dalam linked list
    list.insertDepan(namaAnda, usiaAnda);

    // Data yang dimasukkan sesuai urutan
    list.insertBelakang("John", 19);
    list.insertBelakang("Jane", 20);
    list.insertBelakang("Michael", 18);
    list.insertBelakang("Yusuke", 19);
    list.insertBelakang("Akechi", 20);
    list.insertBelakang("Hoshino", 18);

    // Menampilkan data sebelum operasi
    cout << "Data sebelum operasi: " << endl;
    list.tampilData();
    cout << endl;

    // Hapus data Akechi
    list.hapus("Akechi");

    // Tambahkan data Futaba di antara John dan Jane
    list.insertTengah("Futaba", 18, "John");

    // Tambahkan data Igor di awal
```

```
    list.insertDepan("Igor", 20);

    // Ubah data Michael menjadi Reyn
    list.ubahData("Michael", 18, "Reyn", 18);

    // Menampilkan data setelah operasi
    cout << "Data setelah operasi: " << endl;
    list.tampilData();

    return 0;
}
```

Screenshoot program

```
PS C:\olan\vscode\Modul3> cd "c:\olan\vscode\Modul3\"
Masukkan nama Anda: Maulana Ghani Rolanda
Masukkan usia Anda: 19
Data sebelum operasi:
Maulana Ghani Rolanda   19
John    19
Jane    20
Michael 18
Yusuke  19
Akechi  20
Hoshino 18

Data setelah operasi:
Igor    20
Maulana Ghani Rolanda   19
John    19
Futaba  18
Jane    20
Reyn    18
Yusuke  19
Hoshino 18
PS C:\olan\vscode\Modul3>
```

Deskripsi program

Program tersebut adalah implementasi dari menu Single Linked List Non-Circular dalam bahasa C++. Program ini memungkinkan pengguna untuk memasukkan data mahasiswa (nama dan usia) sesuai dengan operasi yang diinginkan.

1. **Deklarasi Struct Node**: Mendefinisikan sebuah struktur **Node** yang menyimpan data mahasiswa (nama dan usia) serta pointer ke node berikutnya.
2. **Deklarasi Linked List**: Mendefinisikan kelas **LinkedList** yang berisi operasi-operasi untuk mengelola linked list, seperti menambah, menghapus, dan mengubah data, serta menampilkan data.
3. **Metode insertDepan()**: Digunakan untuk menambahkan data mahasiswa di depan linked list.
4. **Metode insertBelakang()**: Digunakan untuk menambahkan data mahasiswa di belakang linked list.
5. **Metode insertTengah()**: Digunakan untuk menambahkan data mahasiswa di tengah linked list setelah node tertentu.
6. **Metode hapus()**: Digunakan untuk menghapus data mahasiswa dari linked list.
7. **Metode ubahData()**: Digunakan untuk mengubah data mahasiswa yang sudah ada di linked list.
8. **Metode tampilData()**: Digunakan untuk menampilkan seluruh data mahasiswa yang ada dalam linked list.
9. **Fungsi main()**: Pengguna memasukkan data mahasiswa pertama kali (nama dan usia mereka). Kemudian, program memasukkan data mahasiswa lain sesuai urutan yang diminta. Setelah itu, program melakukan operasi-operasi yang diminta (penghapusan, penambahan di tengah, penambahan di awal, dan perubahan data) dan menampilkan data sebelum dan sesudah operasi dilakukan.

Program ini mengizinkan pengguna untuk memanipulasi linked list mahasiswa sesuai kebutuhan dengan menggunakan menu interaktif.

2. Unguided 2

Source code

```
#include <iostream>
#include <string>
#include <iomanip>
#include <vector>
using namespace std;

// Struktur data untuk produk skincare
struct Produk {
    string nama;
    int harga;
};

// Kelas untuk toko skincare
class TokoSkincare {
private:
    vector<Produk> produk;

public:
    // Method untuk menambahkan data produk
    void tambahData() {
        Produk p;
        cout << "Masukkan Nama Produk: ";
        getline(cin >> ws, p.nama);
        cout << "Masukkan Harga Produk: ";
        cin >> p.harga;
        produk.push_back(p);
        cout << "Data produk berhasil ditambahkan." << endl;
    }

    // Method untuk menghapus data produk
    void hapusData() {
        if (produk.empty()) {
            cout << "Data produk kosong." << endl;
            return;
        }
        produk.pop_back();
        cout << "Data produk terakhir berhasil dihapus." << endl;
    }
```

```
    // Method untuk mengupdate data produk
    // Method untuk mengupdate data produk
void updateData() {
    if (produk.empty()) {
        cout << "Data produk kosong." << endl;
        return;
    }
    cout << "Masukkan Nama Produk yang Ingin diupdate: ";
    string namaProduk;
    getline(cin >> ws, namaProduk);
    for (int i = 0; i < produk.size(); ++i) {
        if (produk[i].nama == namaProduk) {
            cout << "Masukkan Nama Baru: ";
            getline(cin >> ws, produk[i].nama);
            cout << "Masukkan Harga Baru: ";
            cin >> produk[i].harga;
            cout << "Data produk berhasil diupdate." << endl;
            return;
        }
    }
    cout << "Data produk tidak ditemukan." << endl;
}


    // Method untuk menambahkan data produk di urutan tertentu
    void tambahDataUrutan() {
        Produk p;
        cout << "Masukkan Nama Produk: ";
        getline(cin >> ws, p.nama);
        cout << "Masukkan Harga Produk: ";
        cin >> p.harga;
        cout << "Masukkan Urutan Tempat Penyisipan Data (indeks):
";
        int posisi;
        cin >> posisi;
        if (posisi >= 0 && posisi <= produk.size()) {
            produk.insert(produk.begin() + posisi, p);
            cout << "Data produk berhasil ditambahkan di urutan
tertentu." << endl;
        } else {
```

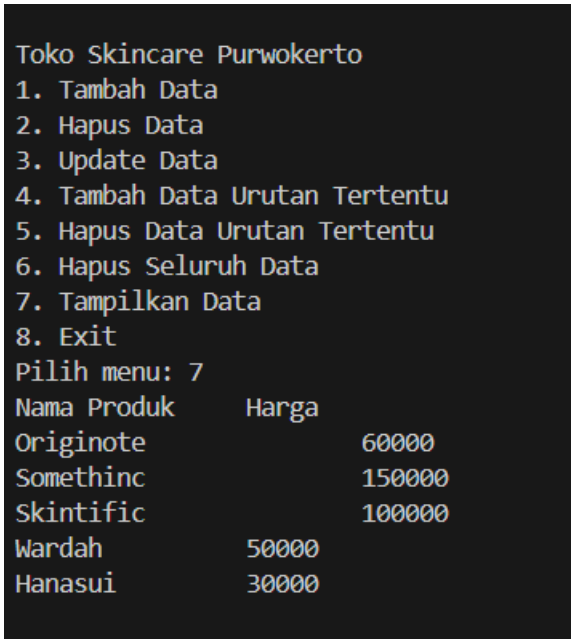
```
            cout << "Posisi tidak valid." << endl;
        }
    }

    // Method untuk menghapus data produk di urutan tertentu
    void hapusDataUrutan() {
        if (produk.empty()) {
            cout << "Data produk kosong." << endl;
            return;
        }
        cout << "Masukkan Urutan Tempat Hapus Data (indeks): ";
        int posisi;
        cin >> posisi;
        if (posisi >= 0 && posisi < produk.size()) {
            produk.erase(produk.begin() + posisi);
            cout << "Data produk pada urutan tertentu berhasil
dihapus." << endl;
        } else {
            cout << "Posisi tidak valid." << endl;
        }
    }

    // Method untuk menghapus seluruh data produk
    void hapusSeluruhData() {
        produk.clear();
        cout << "Seluruh data produk berhasil dihapus." << endl;
    }

    // Method untuk menampilkan seluruh data produk
    void tampilkanData() {
        if (produk.empty()) {
            cout << "Data produk kosong." << endl;
            return;
        }
        cout << "Nama Produk\tHarga" << endl;
        for (int i = 0; i < produk.size(); ++i) {
            cout << produk[i].nama << "\t\t" << produk[i].harga
<< endl;
        }
    }
};
```

```cpp
int main() {
    TokoSkincare toko;
    int choice;

    do {
        cout << "\nToko Skincare Purwokerto" << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
        cout << "3. Update Data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
        cout << "5. Hapus Data Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data" << endl;
        cout << "7. Tampilkan Data" << endl;
        cout << "8. Exit" << endl;
        cout << "Pilih menu: ";
        cin >> choice;
        cin.ignore(); // Membersihkan buffer input sebelumnya

        switch (choice) {
            case 1:
                toko.tambahData();
                break;
            case 2:
                toko.hapusData();
                break;
            case 3:
                toko.updateData();
                break;
            case 4:
                toko.tambahDataUrutan();
                break;
            case 5:
                toko.hapusDataUrutan();
                break;
            case 6:
                toko.hapusSeluruhData();
                break;
            case 7:
                toko.tampilkanData();
                break;
```

```
            case 8:
                cout << "Terima kasih telah menggunakan layanan
toko skincare Purwokerto. Sampai jumpa lagi!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid. Silakan coba lagi."
<< endl;
        }
    } while (choice != 8);

    return 0;
}
```

Screenshoot program

Data sebelum dirubah



Data Setelah Dirubah

```
Toko Skincare Purwokerto
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih menu: 7
Nama Produk     Harga
Originote               60000
Somethinc               150000
Azarine         65000
Skintific               100000
Cleora          55000
```

Deskripsi program

Program di atas adalah sebuah aplikasi sederhana untuk manajemen data produk skincare di sebuah toko di Purwokerto

1) **Struktur Produk**: Mendefinisikan sebuah struct `Produk` yang berisi dua atribut, yaitu `nama` (string) dan `harga` (int). Atribut ini merepresentasikan nama dan harga produk skincare.

2) **Kelas TokoSkincare**: Mendefinisikan sebuah kelas `TokoSkincare` yang memiliki atribut berupa vector dari `Produk`. Kelas ini menyediakan berbagai method untuk menambah, menghapus, mengupdate, dan menampilkan data produk.

3) **Menu Utama**: Program memiliki menu utama yang ditampilkan kepada pengguna. Pengguna dapat memilih berbagai opsi menu untuk

mengelola data produk, seperti menambah, menghapus, mengupdate, atau menampilkan data produk.

4) **Implementasi Opsi Menu**: Setiap opsi menu pada menu utama diimplementasikan dengan memanggil method yang sesuai dari kelas `TokoSkincare`. Misalnya, jika pengguna memilih opsi untuk menambah data, maka program akan memanggil method `tambahData()` dari kelas `TokoSkincare`.

5) **Input dan Output**: Program menggunakan input dari pengguna untuk menambah, menghapus, atau mengupdate data produk. Output dari program berupa pesan yang memberikan informasi tentang keberhasilan operasi yang dilakukan atau pesan kesalahan jika operasi gagal.

6) **Looping**: Program menggunakan loop `do-while` untuk terus menampilkan menu utama dan menerima input dari pengguna hingga pengguna memilih untuk keluar dari program (menu nomor 8).

BAB IV

KESIMPULAN

Pada praktikum kali ini membahas implementasi linked list dalam bahasa pemrograman C++ dengan fokus pada single linked list dan doubly linked list.

1. Konsep Single Linked List dan Doubly Linked List: - Single Linked List adalah struktur data berisi node-node yang saling terhubung satu arah. Setiap node memiliki data dan pointer ke node berikutnya.
- Doubly Linked List adalah struktur data serupa dengan Single Linked List, namun setiap node memiliki pointer ke node sebelumnya dan node berikutnya.

2. Implementasi Single Linked List: - Program implementasi Single Linked List mengizinkan operasi penambahan, penghapusan, perubahan, dan penampilan data.

- Fungsi-fungsi dasar seperti tambah di depan/belakang, di tengah, hapus di depan/belakang/tengah, dan ubah data sudah diimplementasikan.

3. Implementasi Doubly Linked List: - Program implementasi Doubly Linked List juga mengizinkan operasi penambahan, penghapusan, perubahan, dan penampilan data.
- Metode-metode seperti push, pop, update, deleteAll, dan display sudah diimplementasikan.
- Program menggunakan menu interaktif untuk mengelola data produk skincare dalam sebuah toko.
- Pengguna dapat menambah, menghapus, mengupdate, menambahkan data di urutan tertentu, menghapus data di urutan tertentu, menghapus seluruh data, dan menampilkan data.

Dengan menggunakan linked list, kita dapat mengelola data secara dinamis dengan mudah, seperti pada program-program yang dijelaskan dalam laporan praktikum ini.

DAFTAR PUSTAKA

Geeks For Geeks 28 Mar, 2024. Linked List Data Structure, Diakses pada 2 April 2024 dari https://www.geeksforgeeks.org/data-structures/linked-list/?ref=shm