LAPORAN PRAKTIKUM

MODUL III
SINGLE AND DOUBLE LINKED LIST

Disusun oleh:
FATTAH RIZQY ADHIPRATAMA
NIM: 2311102019

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng.

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2024

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

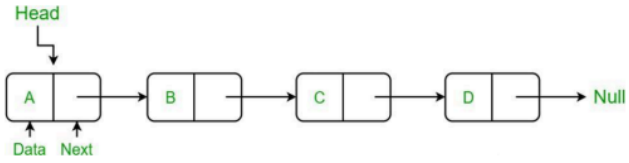
Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis adalah senarai berkait (linked list).Suatu senarai berkait (linked list) adalah suatu simpul (node) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau class. Simpul harus mempunyai satu atau lebih elemen struktur atau class yang berisi data. Secara teori, linked list adalah sejumlah node yang dihubungkan secara linier dengan bantuan pointer. Dikatakan single linked apabila hanya ada satu pointer yang menghubungkan setiap node single.

a. Single Linked List

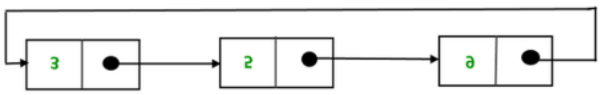
Single Linked list adalah Daftar terhubung yang setiap simpul pembentuknya mempunyai satu rantai(link) ke simpul lainnya. Pembentukan linked list tunggal memerlukan :

1. Deklarasi tipe simpul
2. Deklarasi variabel pointer penunjuk awal Linked list
3. Pembentukan simpul baru
4. Pengaitan simpul baru ke Linked list yang telah terbentuk

Ada beberapa operasi yang dapat dibuat pada senarai tersebut, diantaranya: tambah, hapus dan edit dari senarai tersebut.



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 (DLL) adalah suatu cara pengolahan data yang bekerja dengan record dalam jumlah besar, sehingga membutuhkan alokasi memori dinamis yang besar pula. DLL biasanya digunakan pada saat alokasi memori konvensional tidak lagi bisa diandalkan. Sedangkan bekerja dengan data yang besar tidak dapat dihindari lagi, karena tidak jarang pula, data besar tersebut memiliki hubungan yang erat. Di dalam DLL tidak hanya sekadar menampilkan setiap record-nya, melainkan dapat pula menambahkan record, menghapus beberapa record sesuai keinginan pengguna, sampai mengurutkan record.

Kondisi tersebut memungkinkan dimilikinya satu rantai data yang panjang dan saling berhubungan. Pada Double Linked List, setiap node memiliki dua buah pointer ke sebelah kiri (prev) dan ke sebelah kanan (next). Bertambah lagi komponen yang akan digunakan. Apabila dalam Single Linked List hanya memiliki head, curr dan node, maka untuk Double Linked List, ada satu penunjuk yang berfungsi sebagai akhir dari list: tail. Bagian kiri dari head akan menunjuk ke NULL. Demikian pula dengan bagian kanan dari tail. Setiap node saling terhubung dengan pointer kanan dan kiri.

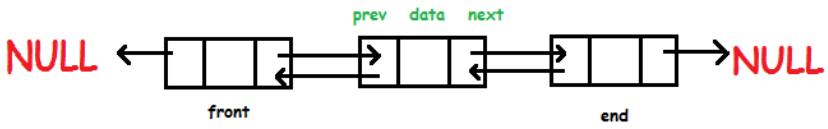
Operasi Double Linked List

1. Menambah sebuah node pada double linked list.
2. Menghapus sebuah node dari double linked list.
3. Mencari node pada double linked list.
4. List tranversal

Struktur Double Linked List

1. Node-node *double linked list* saling berkait melalui pointer.
2. Bagian left sebuah node menunjuk node selanjutnya. Bagian right sebuah node menunjuk node sesudahnya.
3. *pHead* : pointer yang menunjuk node pertama
4. Setiap node terdiri atas
5. Left, yaitu pointer yang menunjuk ke node sebelumnya pada list
6. Data Left, yaitu pointer yang menunjuk ke node sebelumnya pada list
7. Left node pertama bernilai NULL

8. Right node terakhir bernilai NULL



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 D:\Data Semester 2\Praktikum Struktur Data dan Algoritme\Modul 3>
e\Modul 3\" ; if ($?) { g++ guided1.cpp -o guided1 } ; if ($?) { .\gu
3       satu
3       satu    5       dua
2       tiga    3       satu    5       dua
1       empat   2       tiga    3       satu    5       dua
2       tiga    3       satu    5       dua
2       tiga    3       satu
2       tiga    7       sepuluh 3       satu
2       tiga    3       satu
1       delapan 3       satu
1       delapan 8       sembilan
1       delapan 11      tujuh
PS D:\Data Semester 2\Praktikum Struktur Data dan Algoritme\Modul 3>
```

Deskripsi program

Program di atas merupakan implementasi struktur data Linked List Non-Circular (Single Linked List) dalam bahasa pemrograman C++. Struktur “Node” yang memiliki tiga anggota, yaitu “data” (integer), “kata” (string), dan “next”. Terdapat fungsi “init()” untuk menginisialisasi “head” dan “tail” dengan “NULL”, serta fungsi isEmpty() untuk memeriksa apakah linked list kosong atau tidak.

2. Guided 2

Source Code

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

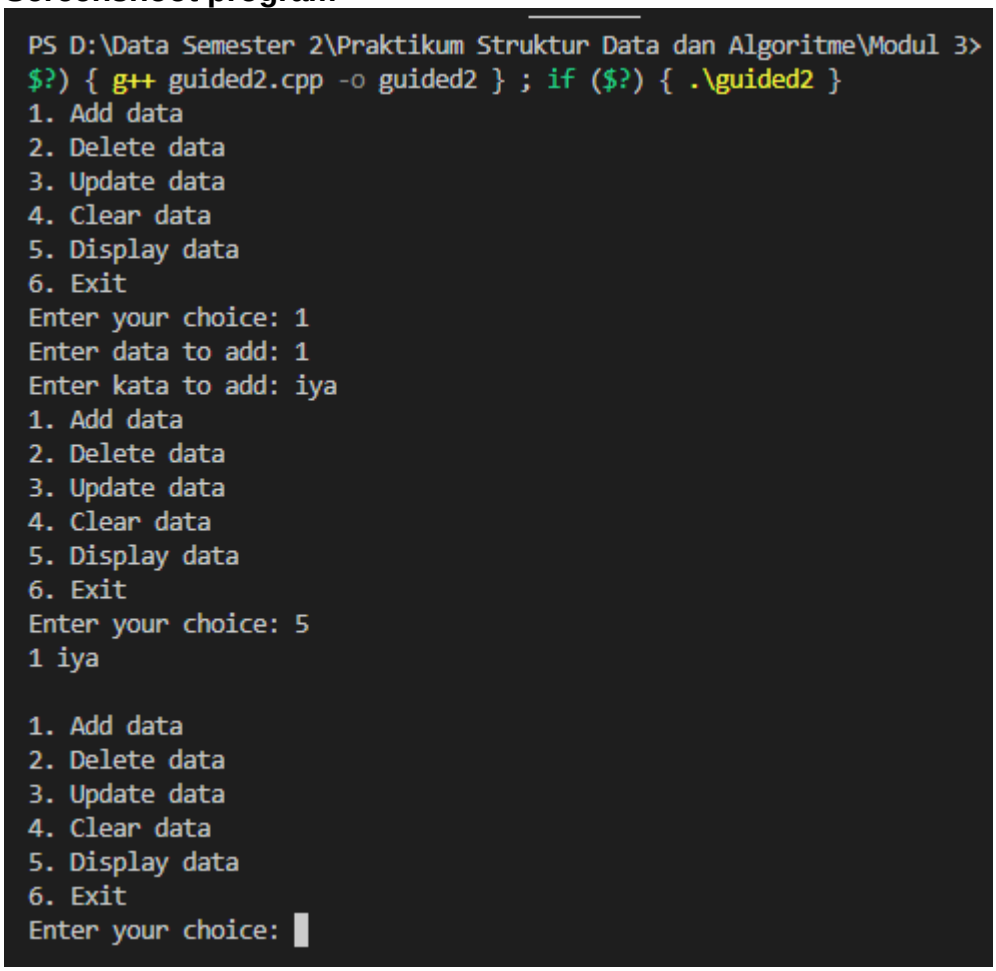
```
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void push(int data, string kata)
    {
        Node *newNode = new Node;
        newNode->data = data;
        newNode->kata = kata;
        newNode->prev = nullptr;
        newNode->next = head;
        if (head != 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;
}
```

Screenshoot program



Deskripsi Program

Program tersebut merupakan implementasi dari sebuah Doubly Linked List dalam bahasa pemrograman C++. Yang meminta pengguna untuk melakukan operasi seperti menambah, menghapus, memperbarui, menghapus semua data, dan menampilkan data yang disimpan dalam list. Node digunakan untuk merepresentasikan simpul dalam Doubly Linked List. Setiap simpul memiliki dua bagian data yaitu integer “data” dan string “kata”, serta dua pointer yaitu “prev” yang menunjuk ke simpul sebelumnya, dan “next” yang menunjuk ke simpul berikutnya.Double Linked List menyediakan berbagai fungsi untuk menjalankan program termasuk:

- “push(int data, string kata)”: Menambahkan simpul baru di awal list.

- “pop()”: Menghapus simpul dari awal list.

- “update(int oldData, int newData, string oldKata, string newKata)”: Memperbarui data dan kata tertentu dalam list.

- deleteAll(): Menghapus semua simpul dari list.
- display(): Menampilkan semua data dan kata dalam list.

LATIHAN KELAS - UNGUIDED

1. **Unguided 1**

   Source code

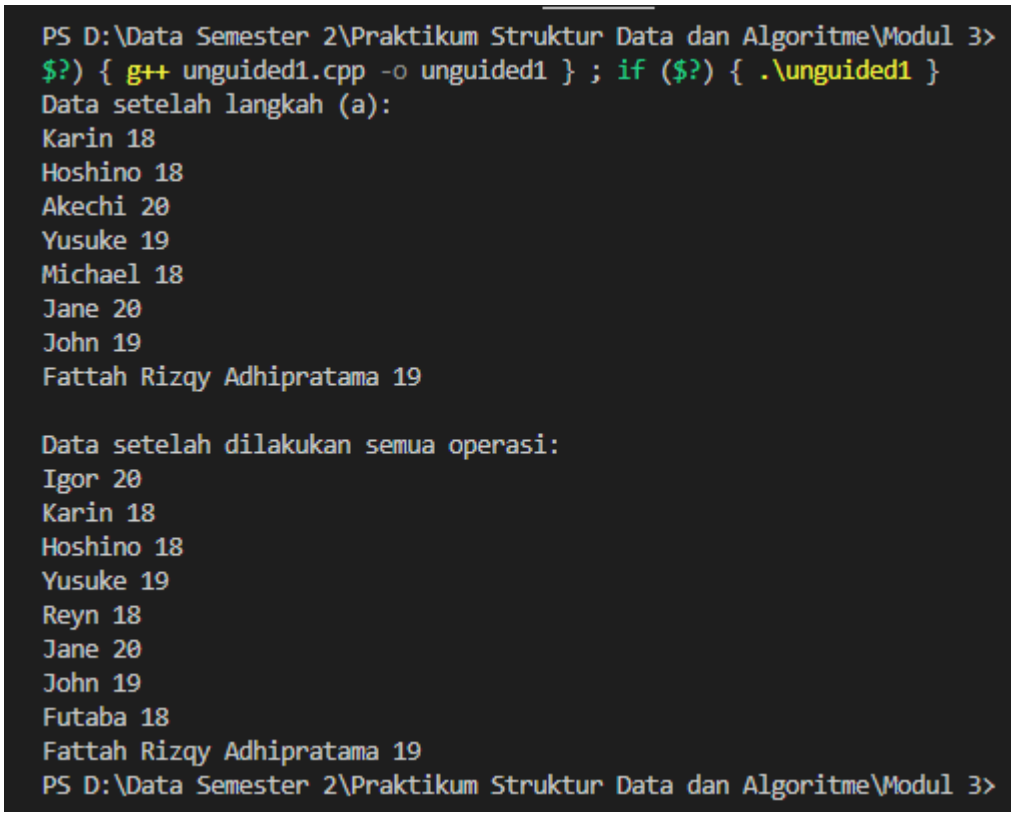
```cpp
#include <iostream>
using namespace std;

struct Node {
    string nama;
    int usia;
    Node* next;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    void insertAwal(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = head;
        head = newNode;
    }

    void insertAkhir(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            return;
```

```
        }

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

    void insertSetelah(string nama, int usia, string namaSebelum) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;

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

        if (temp == nullptr) {
            cout << "Node dengan nama " << namaSebelum << " tidak
ditemukan." << endl;
            return;
        }

        newNode->next = temp->next;
        temp->next = newNode;
    }

    void hapus(string nama) {
        if (head == nullptr) {
            cout << "Linked list kosong." << endl;
            return;
        }

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

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

        if (temp == nullptr) {
            cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
            return;
        }

        prev->next = temp->next;
        delete temp;
    }

    void ubah(string nama, string namaBaru, int usiaBaru) {
        Node* temp = head;
        while (temp != nullptr && temp->nama != nama) {
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
            return;
        }

        temp->nama = namaBaru;
        temp->usia = usiaBaru;
    }

    void tampilkan() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->nama << " " << temp->usia << endl;
            temp = temp->next;
        }
    }
```

```
};

int main() {
    LinkedList myList;

    myList.insertAwal("Fattah Rizqy Adhipratama", 19);
    myList.insertAwal("John", 19);
    myList.insertAwal("Jane", 20);
    myList.insertAwal("Michael", 18);
    myList.insertAwal("Yusuke", 19);
    myList.insertAwal("Akechi", 20);
    myList.insertAwal("Hoshino", 18);
    myList.insertAwal("Karin", 18);

    cout << "Data setelah langkah (a):" << endl;
    myList.tampilkan();
    cout << endl;

    myList.hapus("Akechi");
    myList.insertSetelah("Futaba", 18, "John");
    myList.insertAwal("Igor", 20);
    myList.ubah("Michael", "Reyn", 18);
    cout << "Data setelah dilakukan semua operasi:" << endl;
    myList.tampilkan();

    return 0;
}
```

Screenshoot program



Deskripsi program

Program ini adalah implementasi sederhana dari struktur data Linked List dalam bahasa pemrograman C++. Struct Node mendefinisikan struktur data untuk setiap node dalam Linked List. Setiap node memiliki tiga anggota data: nama (string): Menyimpan nama seseorang. usia (int): Menyimpan usia seseorang. next (pointer ke Node): Pointer yang menunjuk ke node berikutnya dalam Linked List. Class LinkedList menyediakan implementasi operasi-operasi dasar pada Linked List, seperti: “insertAwal(string nama, int usia)”: Menambahkan node baru di awal Linked List. “insertAkhir(string nama, int usia)”: Menambahkan node baru di akhir Linked List. “insertSetelah(string nama, int usia, string namaSebelum)”: Menambahkan node baru setelah node dengan nama tertentu. “hapus(string nama)”: Menghapus node dengan nama tertentu dari Linked List. “ubah(string

nama, string namaBaru, int usiaBaru)": Mengubah data pada node dengan nama tertentu. "tampilkan()": Menampilkan semua data dalam Linked List.

2. **Unguided 2**

Source Code

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

class Node {
public:
    string namaProduk;
    int harga;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    void push(string namaProduk, int harga) {
        Node* newNode = new Node;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        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(string oldNamaProduk, string newNamaProduk, int
newHarga) {
        Node* current = head;
        while (current != nullptr) {
            if (current->namaProduk == oldNamaProduk) {
                current->namaProduk = newNamaProduk;
                current->harga = newHarga;
                return true;
            }
            current = current->next;
        }
        return false;
    }

    void insertAfter(string namaProduk, int harga, string
afterNamaProduk) {
        Node* newNode = new Node;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        Node* current = head;
        while (current != nullptr) {
            if (current->namaProduk == afterNamaProduk) {
                newNode->prev = current;
                newNode->next = current->next;
                if (current->next != nullptr) {
                    current->next->prev = newNode;
                } else {
                    tail = newNode;
```

```
            }
            current->next = newNode;
            return;
        }
        current = current->next;
    }
}

void deleteAfter(string afterNamaProduk) {
    Node* current = head;
    while (current != nullptr) {
        if (current->namaProduk == afterNamaProduk) {
            Node* temp = current->next;
            if (temp != nullptr) {
                current->next = temp->next;
                if (temp->next != nullptr) {
                    temp->next->prev = current;
                } else {
                    tail = current;
                }
                delete temp;
            }
            return;
        }
        current = current->next;
    }
}

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;
```

```
        cout << "Nama Produk\tHarga" << endl;
        while (current != nullptr) {
            cout << current->namaProduk << "\t\t" << current->harga <<
endl;
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "Toko 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;
        int choice;
        cout << "Masukkan pilihan Anda: ";
        cin >> choice;
        switch (choice) {
            case 1: {
                string namaProduk;
                int harga;
                cout << "Masukkan nama produk: ";
                cin >> namaProduk;
                cout << "Masukkan harga: ";
                cin >> harga;
                list.push(namaProduk, harga);
                break;
            }
            case 2: {
                list.pop();
                break;
            }
            case 3: {
```

```
                string oldNamaProduk, newNamaProduk;
                int newHarga;
                cout << "Masukkan nama produk yang lama: ";
                cin >> oldNamaProduk;
                cout << "Masukkan nama produk yang baru: ";
                cin >> newNamaProduk;
                cout << "Masukkan harga yang baru: ";
                cin >> newHarga;
                bool updated = list.update(oldNamaProduk,
newNamaProduk, newHarga);
                if (!updated) {
                    cout << "Data tidak ditemukan" << endl;
                }
                break;
            }
            case 4: {
                string namaProduk, afterNamaProduk;
                int harga;
                cout << "Masukkan nama produk: ";
                cin >> namaProduk;
                cout << "Masukkan harga: ";
                cin >> harga;
                cout << "Masukkan nama produk setelahnya: ";
                cin >> afterNamaProduk;
                list.insertAfter(namaProduk, harga, afterNamaProduk);
                break;
            }
            case 5: {
                string afterNamaProduk;
                cout << "Masukkan nama produk setelahnya: ";
                cin >> afterNamaProduk;
                list.deleteAfter(afterNamaProduk);
                break;
            }
            case 6: {
                list.deleteAll();
                break;
            }
            case 7: {
                list.display();
                break;
```

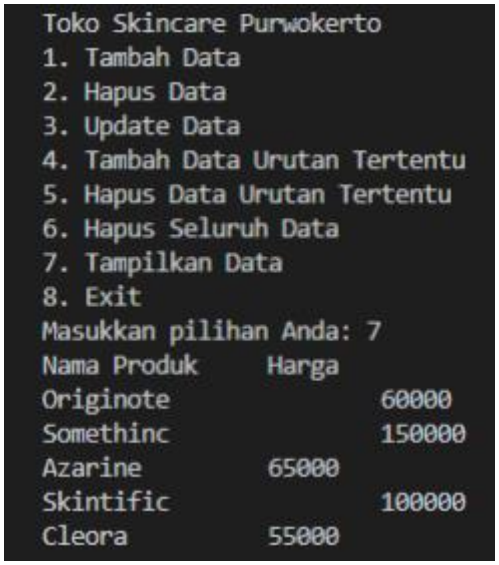
```
            }
            case 8: {
                return 0;
            }
            default: {
                cout << "Pilihan tidak valid" << endl;
                break;
            }
        }
    }
    return 0;
}
```

Screenshoot program

Sebelum

```
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
Masukkan pilihan Anda: 7
Nama Produk     Harga
Originote               60000
Somethinc               150000
Skintific               100000
Wardah          50000
Hanasui         30000
```

Sesudah



Deskripsi Program

Program ini adalah implementasi struktur data Double Linked List dalam bahasa pemrograman C++. Node merepresentasikan setiap node dalam Doubly Linked List. Setiap node memiliki empat anggota data: “namaProduk” (string): Menyimpan nama produk. “harga” (int): Menyimpan harga produk. “prev” (pointer ke Node): Pointer yang menunjuk ke node sebelumnya dalam Double Linked List. “next” (pointer ke Node): Pointer yang menunjuk ke node berikutnya dalam Double Linked List. Double Linked List menyediakan implementasi operasi-operasi pada Double Linked List, seperti: “push(string namaProduk, int harga)”: Menambahkan node baru di awal Double Linked List. “pop()”: Menghapus node di awal Double Linked List. “update(string oldNamaProduk, string newNamaProduk, int newHarga)”: Mengupdate data produk dengan nama tertentu. “insertAfter(string namaProduk, int harga, string afterNamaProduk)”: Menambahkan node baru setelah node dengan nama produk tertentu. “deleteAfter(string afterNamaProduk)”: Menghapus node setelah node dengan nama produk tertentu. “deleteAll()”: Menghapus semua node dalam Double Linked List. “display()”: Menampilkan semua data produk dalam Double Linked List. Program akan terus berjalan hingga pengguna memilih opsi untuk keluar (opsi 8). Setiap kali pengguna memilih opsi, program akan meminta input yang diperlukan (seperti nama produk, harga, atau nama produk setelahnya) dan melakukan operasi yang sesuai pada Double Linked List.

BAB IV

KESIMPULAN

Single Linked List terdiri dari kumpulan node yang terhubung secara linear, di mana setiap node memiliki data dan pointer yang menunjuk ke node berikutnya. Kelebihan single linked list yaitu alokasi memori dinamis dan efisien dalam operasi penambahan dan penghapusan mode, Sedangkan kekurangannya yaitu pencarian node memerlukan traversal dari awal dan tidak dapat melakukan traversal mundur secara efisien.

Double Linked List mirip dengan Single Linked List, namun setiap node memiliki dua pointer, yaitu pointer ke node sebelumnya dan pointer ke node berikutnya. Double Linked list selalu memiliki pointer petunjuk yang selalu menunjuk pada awal dari list yang disebut Head. Double Linked list juga selalu memiliki pointer petunjuk menunjuk pada akhir dari list yang disebut Tail. Double Linked List merupakan salah satu cara untuk mengatasi kelemahan single linked list yang hanya dapat bergerak dalam satu arah saja.

DAFTAR PUSTAKA

Triase. 2020. Diktat Struktur Data. Diakses pada tanggal 31 Maret 2024, dari http://repository.uinsu.ac.id/9717/2/Diktat%20Struktur%20Data.pdf

Syukriyawati, Mahargiyak, Azeri. 2012. Algoritma dan Struktur Data Double Linked List. Diakses pada tanggal 31 Maret 2024, dari https://gusnias.blogspot.com/2013/06/double-linked-list.html