Telkom University Purwokerto

Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Penganalan Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar
13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md

    Sebagai contoh:

    STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

LAPORAN PRAKTIKUM
MODUL 4
SINGLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :

Izzaty Zahara Br Barus – 2311104052

Kelas :

SE-07-02

Dosen :

Wahyu Andi Saputra, S.pd,M.Eng

PROGRAM STUDI SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

I. TUJUAN

1. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada

II. LANDASAN TEORI

Linked list adalah salah satu bentuk struktur data yang terdiri dari elemen-elemen data yang saling terhubung menggunakan pointer. Struktur ini bersifat fleksibel karena dapat bertambah dan berkurang sesuai dengan kebutuhan, tidak seperti array yang bersifat statis. Data yang disimpan dalam linked list bisa berupa data tunggal, seperti nama, atau data majemuk, seperti data mahasiswa yang mencakup nama, NIM, dan alamat.

Kelebihan Linked List dibandingkan Array:

1. Pointer Dinamis: Linked list menggunakan pointer yang bersifat dinamis, sehingga dapat berubah ukuran tanpa harus mengalokasikan memori tambahan seperti pada array yang bersifat statis.
2. Struktur yang Saling Terkait: Elemen-elemen linked list saling terhubung secara langsung, sehingga lebih mudah dikelola dengan pointer.
3. Fleksibilitas: Sifat fleksibel dari linked list sangat sesuai dengan pointer yang dapat diatur sesuai kebutuhan.
4. Efisiensi dalam Pengelolaan Memori: Array cocok untuk data dengan jumlah elemen yang sudah diketahui sejak awal, sedangkan linked list tidak memerlukan ukuran awal yang tetap.

Jenis-jenis Linked List:

1. Single Linked List: Setiap elemen terhubung ke elemen berikutnya dengan pointer tunggal.
2. Double Linked List: Setiap elemen terhubung ke elemen berikutnya dan sebelumnya dengan dua pointer.
3. Circular Linked List: Elemen terakhir terhubung kembali keelemen pertama, membentuk lingkaran.
4. Multi Linked List: Memungkinkan setiap elemen terhubung ke beberapa elemen lainnya dalam struktur yang lebih kompleks.

5. Stack dan Queue: Implementasi struktur data dengan model linked list, di mana stack mengikuti prinsip LIFO (Last In First Out) dan queue mengikuti prinsip FIFO (First In First Out).
6. Tree dan Graph: Merupakan struktur data yang lebih kompleks dengan hubungan antar node yang beragam.

Operasi Dasar pada Linked List:

1. Penciptaan dan Inisialisasi (Create List): Membuat linked list baru dan menginisialisasi elemen pertama (head) menjadi NULL.
2. Penyisipan Elemen (Insert): Menambahkan elemen baru ke awal, akhir, atau setelah elemen tertentu.
3. Penghapusan Elemen (Delete): Menghapus elemen dari linked list.
4. Penelusuran Elemen (View): Menelusuri dan menampilkan seluruh elemen dari linked list.
5. Pencarian Elemen (Search): Mencari elemen tertentu dalam linked list.
6. Pengubahan Isi Elemen (Update): Mengubah data dalam elemen tertentu setelah ditemukan melalui pencarian.

Single Linked List adalah salah satu jenis linked list yang hanya menggunakan satu arah pointer. Setiap elemen dalam linked list ini terdiri dari dua bagian utama:

1. Data: Bagian yang menyimpan informasi utama.
2. Pointer ke Suksesor (Next): Pointer yang menunjukkan elemen berikutnya.

Sifat-sifat dari single linked list antara lain:

- Memiliki pointer yang menunjuk ke elemen berikutnya, dan elemen terakhir menunjuk ke NULL.
- Hanya memungkinkan pembacaan maju, dari elemen pertama hingga elemen terakhir.
- Lebih mudah melakukan penyisipan atau penghapusan elemen di tengah list karena hanya perlu memperbarui pointer.

III. GUIDE

1. Guide1
   a. Syntax

```
#include <iostream>
#include <cstring>
#include <iomanip>
```

```
using namespace std;

struct mahasiswa {
char nama[30];
char nim[10];
};

struct Node {
mahasiswa data;
Node *next;
};

Node *head;
Node *tail;

void init() {
head = nullptr;
tail = nullptr;
}

bool isEmpty() {
return head == nullptr;
}

void insertDepan(const mahasiswa &data) {
Node *baru = new Node;
baru->data = data;
baru->next = nullptr;
if (isEmpty()) {
head = tail = baru;
} else {
baru->next = head;
head = baru;
}
}

void insertBelakang(const mahasiswa &data) {
```

```
Node *baru = new Node;
baru->data = data;
baru->next = nullptr;
if (isEmpty()) {
head = tail = baru;
} else {
tail->next = baru;
tail = baru;
}
}

int hitungList() {
Node *current = head;
int jumlah = 0;
while (current != nullptr) {
jumlah++;
current = current->next;
}
return jumlah;
}

void hapusDepan() {
if (!isEmpty()) {
Node *hapus = head;
head = head->next;
delete hapus;
if (head == nullptr) {
tail = nullptr;
}
} else {
cout << "List Kosong!!" << endl;
}
}

void hapusBelakang() {
if (!isEmpty()) {
if (head == tail) {
delete head;
```

```
head = tail = nullptr;
} else {
Node *bantu = head;
while (bantu->next != tail) {
bantu = bantu->next;
}
delete tail;
tail = bantu;
tail->next = nullptr;
}
} else {
cout << "List kosong!" << endl;
}
}

void tampil() {
Node *current = head;
if (!isEmpty()) {
cout << "Daftar Mahasiswa:" << endl;
cout << setw(30) << left << "Nama" << setw(10) << left << "NIM" << endl;
cout << string(40, '-') << endl;
while (current != nullptr) {
cout << setw(30) << left << current->data.nama << setw(10) << left << current->data.nim
<< endl;
current = current->next;
}
} else {
cout << "List masih kosong!" << endl;
}
}

void clearList() {
Node *current = head;
while (current != nullptr) {
Node *hapus = current;
current = current->next;
delete hapus;
}
```

```
head = tail = nullptr;
cout << "List berhasil terhapus!" << endl;
}

int main() {
init();
mahasiswa m1 = {"Alice", "123456"};
mahasiswa m2 = {"Bob", "654321"};
mahasiswa m3 = {"Charlie", "112233"};

insertDepan(m1);
tampil();
insertBelakang(m2);
tampil();
insertDepan(m3);
tampil();

hapusDepan();
tampil();
hapusBelakang();
tampil();

clearList();
return 0;
}
```

b. Penjelasan

Struktur Data:

struct mahasiswa: Mendefinisikan struktur data mahasiswa yang berisi dua atribut:

nama: Menyimpan nama mahasiswa (tipe char[30]).

nim: Menyimpan NIM mahasiswa (tipe char[10]).

struct Node: Mendefinisikan struktur data node untuk Linked List yang berisi dua atribut:

data: Berisi informasi mahasiswa yang disimpan menggunakan tipe mahasiswa.

next: Pointer yang menghubungkan node ini dengan node berikutnya.

2. Inisialisasi Node:

Node *head dan Node *tail: Pointer head menunjuk ke node pertama dari linked list, sedangkan tail menunjuk ke node terakhir.

3. Fungsi init():

Menginisialisasi linked list dengan membuat head dan tail bernilai nullptr, yang berarti list kosong.

4. Fungsi isEmpty():

Mengecek apakah list kosong. Jika head adalah nullptr, maka list dianggap kosong dan fungsi mengembalikan true.

5. Fungsi insertDepan():

Menambahkan data mahasiswa di depan linked list. Jika list kosong, node baru akan menjadi head dan tail. Jika tidak, node baru akan dihubungkan di depan node yang sudah ada.

6. Fungsi insertBelakang():

Menambahkan data mahasiswa di belakang linked list. Jika list kosong, node baru akan menjadi head dan tail. Jika tidak, node baru dihubungkan di belakang node terakhir (tail).

7. Fungsi hitungList():

Menghitung jumlah elemen (node) dalam linked list dengan melakukan iterasi dari head hingga nullptr.

8. Fungsi hapusDepan():

Menghapus node pertama (depan) dari linked list. Jika hanya ada satu elemen, head dan tail akan menjadi nullptr.

9. Fungsi hapusBelakang():

Menghapus node terakhir dari linked list. Jika hanya ada satu elemen, head dan tail akan menjadi nullptr. Jika tidak, node terakhir dihapus, dan node kedua terakhir menjadi node tail.

10. Fungsi tampil():

Menampilkan data mahasiswa dari seluruh node dalam list. List ditelusuri dari head hingga tail, dan setiap data ditampilkan dalam format tabel.

11. Fungsi clearList():

Menghapus seluruh node dari linked list dengan melakukan iterasi dari head hingga list kosong, lalu menghapus memori yang digunakan.

12. Fungsi main():

Inisialisasi list dengan memanggil fungsi init().

Beberapa mahasiswa (m1, m2, m3) ditambahkan ke linked list dengan fungsi insertDepan() dan insertBelakang().

List ditampilkan setelah setiap operasi penambahan.

Node di depan dan di belakang list dihapus menggunakan hapusDepan() dan hapusBelakang().

List dihapus sepenuhnya dengan clearList(), dan list kosong ditampilkan.

Secara keseluruhan, kode ini menunjukkan bagaimana linked list diimplementasikan dengan operasi dasar seperti penyisipan di depan/belakang, penghapusan di depan/belakang, dan penelusuran list.

c. Output

```
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Nama                            NIM
---------------------------------------
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Charlie                         112233
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
Bob                             654321
Daftar Mahasiswa:
Nama                            NIM
---------------------------------------
Alice                           123456
List berhasil terhapus!
```

2. GUIDE 2

a. Syntax

```cpp
#include <iostream>

using namespace std;

struct Node
{
    int data;
    Node* next;
};

Node* alokasi(int value) {
    Node* newNode = new Node;
    if (newNode != nullptr) {
        newNode->data = value;
```

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

void dealokasi(Node* node) {
    delete node;
}

bool isListEmpty(Node* head) {
    return head == nullptr;
}

void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value);
    if (newNode != nullptr) {
        newNode->next = head;
        head = newNode;
    }
}

void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value);
    if (newNode != nullptr) {
        if (isListEmpty(head)) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
}

void printList(Node* head) {
    if (isListEmpty(head)) {
```

```
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }
}

int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

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

int main() {
    Node* head = nullptr;

    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
    insertLast(head, 30);  // Menambahkan elemen 30 di akhir list

    cout << "Isi List: ";
    printList(head);
```

```
    cout << "Jumlah elemen: " << countElements(head) << endl;

    clearList(head);

    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

b. Penjelasan

Kode di atas adalah implementasi dasar dari Single Linked List dalam bahasa C++. Berikut adalah penjelasan singkatnya:

Struktur Node:

struct Node: Mendefinisikan elemen (node) dalam linked list, yang berisi dua bagian:

data: Menyimpan nilai dari tipe integer.

next: Pointer yang menunjuk ke node berikutnya dalam list.

Fungsi Utama:

Node* alokasi(int value):Mengalokasikan memori untuk node baru dan menginisialisasi nilai data serta mengatur pointer next ke nullptr.

void dealokasi(Node* node):Menghapus (dealokasi) node dari memori.

bool isListEmpty(Node* head):Mengecek apakah linked list kosong. Mengembalikan true jika pointer head adalah nullptr (list kosong).

void insertFirst(Node* &head, int value): Menambahkan node baru di awal linked list. Node baru ini akan menjadi node pertama, dan head akan menunjuk ke node tersebut.

void insertLast(Node* &head, int value): Menambahkan node baru di akhir linked list. Jika list kosong, node baru menjadi elemen pertama. Jika tidak, node baru ditambahkan setelah node terakhir.

void printList(Node* head): Menampilkan semua elemen dalam linked list. Jika list kosong, menampilkan pesan "List kosong!".

int countElements(Node* head): Menghitung jumlah elemen (node) dalam linked list dengan cara menelusuri semua node dan menghitungnya satu per satu.

void clearList(Node* &head): Menghapus seluruh elemen dalam linked list satu per satu, dan membebaskan memori yang digunakan oleh node-node tersebut.

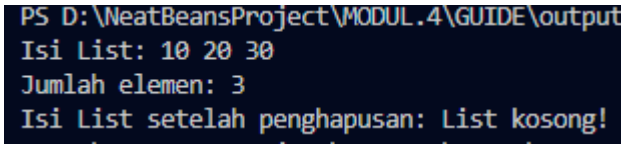
Fungsi main(): Inisialisasi: Pointer head diatur menjadi nullptr, yang menunjukkan bahwa list kosong.

Insert: Menambahkan elemen 10 di awal list. Menambahkan elemen 20 dan 30 di akhir list.

Menampilkan: Menampilkan isi list menggunakan printList(). Menghitung dan menampilkan jumlah elemen dalam list menggunakan countElements().

Clear:Menghapus semua elemen dari list menggunakan clearList(). Setelah penghapusan, isi list ditampilkan kembali untuk menunjukkan bahwa list kosong.

c. Output



IV. UNGUIDED

1. TASK 1

a. Syntax

```
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
};

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() {
        head = nullptr;
    }
    void insertAtFront(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = head;
        head = newNode;
    }
    void insertAtBack(int value) {
        Node* newNode = new Node();
        newNode->data = value;
        newNode->next = nullptr;

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

```
            }
            temp->next = newNode;
        }
    }
    void printList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }

    ~SingleLinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main() {
    SingleLinkedList list;
    list.insertAtFront(10);
    list.insertAtBack(20);
    list.insertAtFront(5);
    cout << "Isi Linked List: ";
    list.printList();

    return 0;
}
```

b. Penjelasan

Kode di atas adalah implementasi Single Linked List menggunakan kelas (class) di C++. Berikut adalah penjelasan singkatnya:

Struktur Node:

struct Node: Mendefinisikan elemen (node) dalam linked list, yang memiliki:

data: Menyimpan nilai integer.

next: Pointer yang menunjuk ke node berikutnya dalam linked list.

Kelas SingleLinkedList:

Private Attribute:Node* head: Pointer yang menunjuk ke node pertama dalam linked list. Jika head adalah nullptr, maka linked list kosong.

Konstruktor:SingleLinkedList(): Menginisialisasi head sebagai nullptr saat objek list pertama kali dibuat, menandakan bahwa list masih kosong.

Fungsi insertAtFront(int value):Menambahkan node baru di awal linked list. Node baru ini menjadi elemen pertama dan head menunjuk ke node tersebut.

Fungsi insertAtBack(int value):Menambahkan node baru di akhir linked list. Jika list kosong, node baru menjadi elemen pertama. Jika tidak, node baru ditambahkan setelah node terakhir.

Fungsi printList():Menampilkan semua elemen dalam linked list. Elemen-elemen dihubungkan dengan tanda panah " -> " antara setiap node.

Destruktor ~SingleLinkedList():Menghapus seluruh node dalam linked list ketika objek list dihapus. Ini memastikan tidak ada kebocoran memori dengan membebaskan semua memori yang digunakan oleh node.

Fungsi main():Membuat Linked List: Membuat objek list dari kelas SingleLinkedList.

Insert:

Menambahkan elemen 10 di awal list.

Menambahkan elemen 20 di akhir list.

Menambahkan elemen 5 di awal list.

Menampilkan Isi List:

Menampilkan isi linked list dengan elemen-elemen yang terhubung menggunakan tanda " -> ".

c. Output

```
PS D:\NeatBeansProject\MODUL.4\UNGU
Isi Linked List: 5 -> 10 -> 20
PS D:\NeatBeansProject\MODUL.4\UNGU
```

2. TASK 2

a. Syntax

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
};

void addFront(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = head;
```

```
    head = newNode;
}

void addBack(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

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

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

void deleteNode(Node*& head, int value) {
    if (head == nullptr) return;

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

    Node* current = head;
    Node* previous = nullptr;

    while (current != nullptr && current->data != value) {
        previous = current;
        current = current->next;
    }
```

```
    if (current == nullptr) return;

    previous->next = current->next;
    delete current;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;
    addFront(head, 10);
    addBack(head, 20);
    addFront(head, 5);

    cout << "Linked List sebelum penghapusan: ";
    printList(head);

    deleteNode(head, 10);

    cout << "Linked List setelah penghapusan node dengan nilai 10: ";
    printList(head);

    while (head != nullptr) {
        deleteNode(head, head->data);
    }

    return 0;
}
```

b. Penjelasan

Kode di atas mengimplementasikan Single Linked List dengan beberapa fungsi dasar: menambahkan node di depan dan belakang, menghapus node berdasarkan nilai, serta mencetak isi list. Berikut penjelasannya:

Struktur Node:

struct Node: Setiap node dalam linked list menyimpan:

data: Nilai integer.

next: Pointer ke node berikutnya.

Fungsi Utama:

addFront(Node*& head, int value): Menambahkan node baru di depan linked list. Node baru akan menjadi node pertama dan head akan menunjuk ke node tersebut.

addBack(Node*& head, int value): Menambahkan node baru di belakang linked list. Jika list kosong, node baru menjadi elemen pertama. Jika tidak, node baru ditambahkan di akhir list.

deleteNode(Node*& head, int value):

Menghapus node dengan nilai tertentu: Jika node pertama (head) memiliki nilai tersebut, node pertama akan dihapus. Jika tidak, fungsi akan menelusuri list untuk menemukan node dengan nilai tersebut dan menghapusnya.

printList(Node* head):Menampilkan isi linked list dengan menghubungkan elemen-elemen menggunakan tanda " -> ".

Fungsi main():

Membuat Linked List:

Menambahkan elemen 10 di depan list.

Menambahkan elemen 20 di belakang list.

Menambahkan elemen 5 di depan list.

Menampilkan Isi List:Sebelum dan sesudah menghapus node dengan nilai 10.

Menghapus Semua Elemen:Menghapus semua node satu per satu dalam loop hingga list kosong.

c. Output

```
PS D:\NeatBeansProject\MODUL.4\UNGUIDED\output> & .\'TASK2.exe'
Linked List sebelum penghapusan: 5 -> 10 -> 20
Linked List setelah penghapusan node dengan nilai 10: 5 -> 20
PS D:\NeatBeansProject\MODUL.4\UNGUIDED\output>
```

3. TASK 3

a. Syntax

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() : head(nullptr) {}
```

```
void addFront(int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

void addBack(int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

bool search(int value) {
    Node* temp = head;
    while (temp) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

int length() {
    int count = 0;
    Node* temp = head;
    while (temp) {
        count++;
        temp = temp->next;
    }
    return count;
```

```
    }
};

int main() {
    LinkedList list;

    list.addFront(10);
    list.addBack(20);
    list.addFront(5);


    if (list.search(20)) {
        cout << "Node dengan nilai 20 ditemukan." << endl;
    } else {
        cout << "Node dengan nilai 20 tidak ditemukan." << endl;
    }

    cout << "Panjang linked list: " << list.length() << endl;

    return 0;
}
```

b. Penjelasan

Kode di atas mengimplementasikan Single Linked List menggunakan kelas (LinkedList) di C++. Berikut adalah penjelasan singkatnya:

Struktur Node:

struct Node: Setiap node menyimpan:

data: Nilai integer.

next: Pointer ke node berikutnya.

Constructor: Menginisialisasi node dengan nilai val dan mengatur next sebagai nullptr.

Kelas LinkedList:

Atribut Private:Node* head: Pointer yang menunjuk ke node pertama dalam linked list.

Konstruktor LinkedList(): Menginisialisasi linked list dengan head yang bernilai nullptr, menandakan bahwa list kosong.

Fungsi addFront(int value): Menambahkan node baru di depan linked list. Node baru menjadi elemen pertama, dan head akan menunjuk ke node tersebut.

Fungsi addBack(int value): Menambahkan node baru di akhir linked list. Jika list kosong, node baru menjadi elemen pertama. Jika tidak, node baru ditambahkan setelah node terakhir.

Fungsi search(int value): Mencari node dengan nilai tertentu dalam linked list. Jika ditemukan, mengembalikan true, jika tidak, false.

Fungsi length(): Menghitung jumlah elemen (node) dalam linked list dengan menelusuri list dari awal hingga akhir.

Fungsi main():

Membuat Linked List:

Menambahkan elemen 10 di depan list.

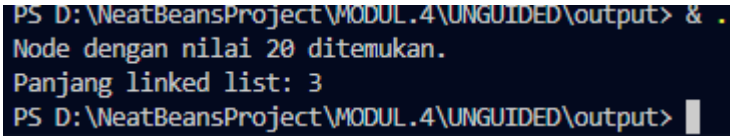
Menambahkan elemen 20 di belakang list.

Menambahkan elemen 5 di depan list.

Pencarian Elemen: Mencari node dengan nilai 20 dalam list, dan menampilkan pesan apakah node tersebut ditemukan.

Menampilkan Panjang List: Menghitung dan menampilkan jumlah elemen dalam linked list.

c. Output



V. KESIMPULAN

Linked List merupakan struktur data yang dinamis di mana elemen-elemen data dihubungkan melalui pointer. Berbeda dengan array yang memiliki ukuran tetap, Linked List memungkinkan penambahan atau pengurangan elemen secara fleksibel. Struktur data ini memiliki berbagai jenis, seperti Single Linked List, Double Linked List, dan Circular Linked List, masing-masing dengan kegunaannya sendiri.

Operasi dasar dalam Linked List meliputi:

- Penyisipan Elemen: Baik di awal (insertDepan) atau akhir (insertBelakang) list.
- Penghapusan Elemen: Dari depan (hapusDepan) atau belakang (hapusBelakang) list.
- Penelusuran dan Pencarian: Menggunakan fungsi untuk menelusuri seluruh elemen atau mencari elemen tertentu dalam list.
- Penghitungan dan Pengosongan: Menghitung jumlah elemen yang ada dalam list, serta mengosongkan list dengan menghapus seluruh elemen.

Beberapa implementasi yang dicontohkan dalam kode di atas memperlihatkan cara penggunaan struktur Linked List, seperti menambahkan, menghapus, dan mencetak elemen-elemen yang ada. Selain itu, implementasi ini juga menunjukkan pentingnya pengelolaan memori melalui alokasi dan dealokasi pointer secara tepat agar tidak terjadi kebocoran memori. Dengan demikian, Linked List adalah solusi yang sangat fleksibel dan efisien untuk mengelola data yang ukuran atau jumlahnya dapat berubah-ubah selama eksekusi program.