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 :

Zaenarif Putra ‘Ainurdin – 2311104049

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

1. Linked List dengan Pointer

   Linked list (biasa disebut list saja) adalah salah satu bentuk struktur data (representasi penyimpanan) berupa serangkaian elemen data yang saling berkait (berhubungan) dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Data yang disimpan dalam Linked list bisa berupa data tunggal atau data majemuk. Data tunggal merupakan data yang hanya terdiri dari satu data (variabel), misalnya: nama bertipe string. Sedangkan data majemuk merupakan sekumpulan data (record) yang di dalamnya terdiri dari berbagai tipe data, misalnya: Data Mahasiswa, terdiri dari Nama bertipe string, NIM bertipe long integer, dan Alamat bertipe string.

2. Single Linked List

   Single Linked List adalah model ADT Linked List yang memiliki arah pointer tunggal, di mana setiap elemen dalam list terdiri dari data dan pointer yang disebut suksesor, yang menghubungkan elemen satu dengan yang lain. Elemen pertama dikenal sebagai head, dan setiap elemen berikutnya memiliki pointer yang mengacu pada elemen selanjutnya. Pointer pada elemen terakhir menunjuk ke nilai Null atau Nil, kecuali pada list circular. List ini hanya mendukung pembacaan maju dan pencarian sequensial jika data tidak terurut. Kelebihan dari Single Linked List adalah kemudahan dalam penyisipan atau penghapusan elemen di tengah list.

3. Pembentukan Komponen-Komponen List

   Pembentukan list melibatkan proses inisialisasi dengan fungsi seperti `createList()`, yang mengatur pointer awal dan akhir list menjadi `Nil`. Pengalokasian memori untuk elemen-elemen list dilakukan dengan menggunakan `new` dalam C++, menggantikan penggunaan `malloc` pada C. Contoh alokasi memori melibatkan pointer yang menunjuk ke elemen yang dialokasikan, seperti data mahasiswa dengan nama dan nim. Dealokasi memori yang telah digunakan dilakukan dengan sintaks `delete` pada C++, menggantikan `free` pada C. Pengecekan apakah list kosong dilakukan dengan fungsi `isEmpty()`, yang mengembalikan nilai true jika list kosong.

4. Insert

   Insert First : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada awal list.
   Insert Last : Merupakan metode memasukkan elemen data ke dalam list yang diletakkan pada akhir list. Insert After : Merupakan metode memasukkan data ke dalam list yang diletakkan setelah node tertentu yang
   ditunjuk oleh user.

5. View

   Merupakan operasi dasar pada list yang menampilkan isi node/simpul dengan suatu penelusuran list. Mengunjungi setiap node kemudian menampilkan data yang tersimpan pada node tersebut.

6. Delete

   Penghapusan elemen dalam linked list mencakup beberapa operasi dasar: Delete First, yang menghapus elemen pertama dari list; Delete Last, yang menghapus elemen terakhir; Delete After, yang menghapus node setelah node tertentu; dan Delete Elemen, yang menghapus elemen tertentu serta membebaskan memori yang digunakan. Fungsi umum yang digunakan dalam penghapusan ini meliputi dealokasi(P) untuk membebaskan memori elemen P, dan delAll(L) untuk membebaskan semua memori elemen dalam list L, sehingga list menjadi kosong. Semua operasi dasar ini dikenal sebagai operasi primitif pada linked list.

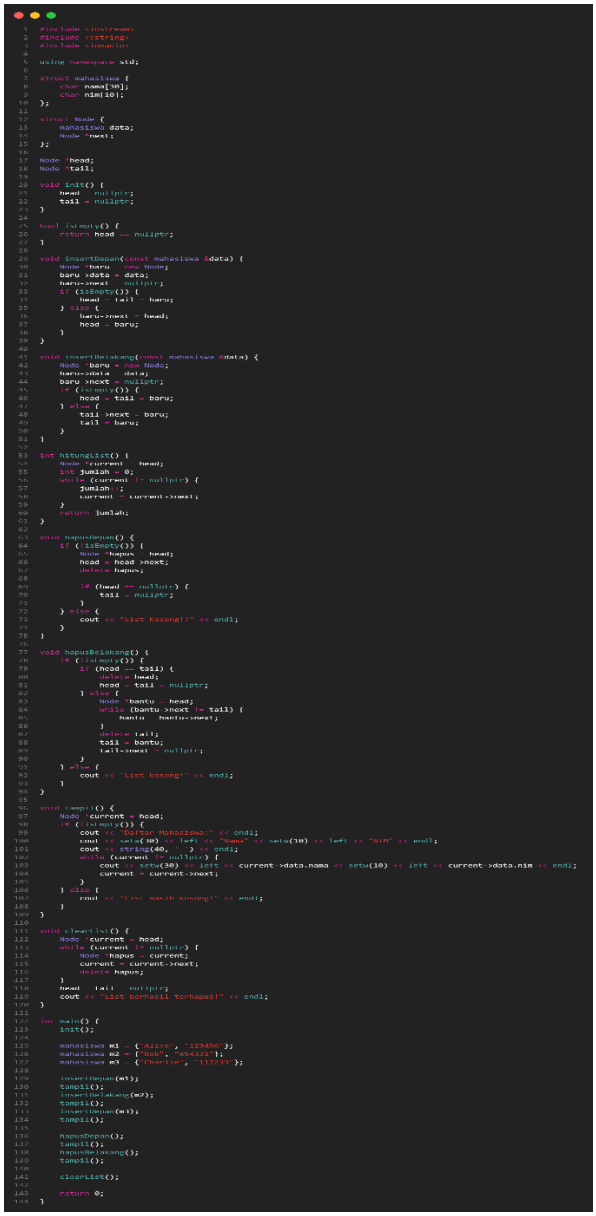
7. Update

   Merupakan operasi dasar pada list yang digunakan untuk mengupdate data yang ada di dalam list. Dengan operasi update ini kita dapat meng-update data-data node yang ada di dalam list. Proses update biasanya diawali dengan proses pencarian terhadap data yang akan di-update.

III. GUIDE

1. Guide_1

   a. Syntax



b. Penjelasan Syntax

Struct mahasiswa: Menyimpan informasi mahasiswa dengan dua atribut: nama (maksimal 30 karakter) dan nim (maksimal 10 karakter).

Struct Node: Merupakan elemen dari linked list yang menyimpan data mahasiswa dan pointer ke node berikutnya.

Pointer head dan tail: head menunjuk ke node pertama dalam linked list sedangkan tail menunjuk ke node terakhir dalam linked list.

void init(): Digunakan untuk menginisialisasi linked list dengan head dan tail diatur ke nullptr.

bool isEmpty(): Digunakan sebagai pengembalian nilai true jika linked list kosong (yaitu, head adalah nullptr).

void insertDepan(const mahasiswa &data):Digunakan untu Menyisipkan node baru di depan linked list. Jika list kosong, node baru menjadi head dan tail. Jika tidak, node baru menjadi head dan pointer next diatur ke node sebelumnya.

void insertBelakang(const mahasiswa &data):Untuk menyisipkan node baru di belakang linked list. Jika list kosong, node baru menjadi head dan tail. Jika tidak, node baru ditambahkan di belakang dan tail diperbarui.

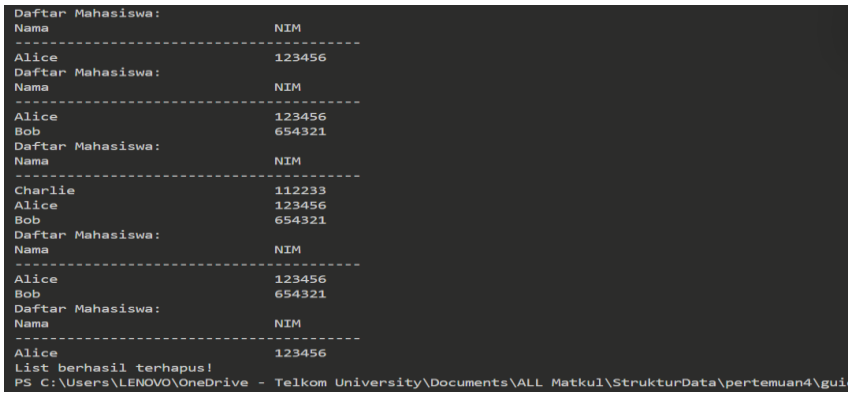
int hitungList(): Menghitung jumlah node dalam linked list dengan mengiterasi dari head hingga nullptr.

void hapusBelakang(): Menghapus node dari belakang linked list. Jika hanya ada satu node, hapus node tersebut dan atur head dan tail ke nullptr.

void tampil(): Menampilkan semua node dalam linked list dengan format yang rapi. void clearList(): Menghapus semua node dalam linked list dan mengatur head dan tail ke nullptr.

Inisialisasi linked list dengan init(), lalu tambahkan objek mahasiswa ke dalamnya menggunakan insertDepan() dan insertBelakang(). Tampilkan isi linked list menggunakan tampil(), kemudian hapus node dari depan dan belakang dengan hapusDepan() serta hapusBelakang(). Setelah itu, tampilkan kembali isi linked list dan terakhir hapus semua node menggunakan clearList().

c. Output



2. Task_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);
    insertLast(head, 20);
    insertLast(head, 30);

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

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

    clearList(head);

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

    return 0;
}
```

b. Penjelasan Syntax

struct Node : Mendefinisikan struktur Node yang akan menjadi elemen dari linked list. Setiap node menyimpan data integer (data) dan pointer ke node berikutnya (next).

Node* alokasi(int value) Fungsi ini mengalokasikan memori untuk node baru. Jika berhasil, node diisi dengan nilai (value) yang
diberikan, dan pointer next diatur menjadi nullptr (karena node baru belum terhubung ke node lain).

void dealokasi(Node* node) : Fungsi ini menghapus node dari memori menggunakan delete.

bool isListEmpty(Node* head) Fungsi ini memeriksa apakah linked list kosong. Jika head adalah nullptr, berarti linked list kosong.

void insertFirst(Node* &head, int value) : Fungsi ini menambahkan node baru di bagian depan linked list. Node baru dibuat dengan nilai yang diberikan dan menjadi head (kepala) dari linked list, sedangkan next node baru dihubungkan ke node lama yang sebelumnya berada di depan.

void insertLast(Node* &head, int value) : Fungsi ini menambahkan node baru di bagian akhir linked list. Jika linked list kosong, node baru menjadi head. Jika tidak, fungsi ini mencari node terakhir dalam list dan menghubungkannya dengan node baru.

void printList(Node* head) : Fungsi ini mencetak semua elemen dalam linked list. Jika list kosong, mencetak "List kosong!". Jika tidak, iterasi dilakukan dari head hingga node terakhir, mencetak setiap elemen.

int countElements(Node* head) : Fungsi ini menghitung dan mengembalikan jumlah elemen (node) dalam linked list dengan menghitung setiap node mulai dari head hingga node terakhir.

void clearList(Node* &head) : Fungsi ini menghapus semua node dari linked list satu per satu, mulai dari head. Setiap node dihapus dari memori dengan dealokasi.

c. Output

```
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\guide\output>

Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\guide\output>
```

IV. UNGUIDED

1. Membuat Single Linked List

   a. Syntax

```cpp
#include <iostream>

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) {
            std::cout << temp->data;
            if (temp->next != nullptr) {
                std::cout << " -> ";
            }
            temp = temp->next;
        }
        std::cout << std::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);

    std::cout << "Output : " << std::endl;
    list.printList();

    return 0;
}
```

b. Penjelasan Syntax

struct Node : Mendefinisikan struktur Node yang akan digunakan dalam linked list. Setiap node memiliki data (int data) dan pointer ke node berikutnya (Node* next).

class SingleLinkedList : Mendefinisikan kelas SingleLinkedList yang berisi beberapa operasi untuk linked list.

private: Node* head; : Bagian private dari kelas yang menyimpan pointer head, yang menunjuk ke node pertama dari linked list. Bagian ini hanya bisa diakses dari dalam kelas.

public: SingleLinkedList() : Konstruktor kelas, yang diinisialisasi saat objek SingleLinkedList dibuat. Di sini, head diatur menjadi nullptr, artinya linked list dimulai dalam keadaan kosong.

void insertAtFront(int value) : Fungsi ini menambahkan node baru di bagian depan linked list dengan membuat node baru (newNode), mengisi data node tersebut dengan nilai value, menghubungkannya ke head, lalu mengatur head menjadi newNode sehingga node baru menjadi node pertama di list.

void insertAtBack(int value) : Fungsi ini menambahkan node baru di bagian belakang linked list dengan membuat node baru (newNode) dan mengisi datanya; jika head adalah nullptr (list kosong), node baru menjadi node pertama, sedangkan jika list tidak kosong, fungsi ini berjalan hingga node terakhir dan menambahkan node baru di akhir list.

~SingleLinkedList() :Destruktor yang dipanggil ketika objek SingleLinkedList dihancurkan (misalnya, ketika program selesai):
Menghapus semua node dalam linked list untuk menghindari kebocoran memori. Node dihapus satu per satu mulai dari head hingga node terakhir.

int main() : Membuat objek SingleLinkedList bernama list. Menambahkan beberapa elemen ke linked list menggunakan insertAtFront() dan insertAtBack(). Mencetak isi linked list menggunakan printList(). Ketika program selesai, destruktor ~SingleLinkedList() akan otomatis memanggil fungsi untuk membersihkan memori.

c. Output

```
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\unguided\output>
e'
Output :
5 -> 10 -> 20
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\unguided\output>
```

2. Menghapus Node pada Linked List

a. Syntax

```cpp
#include <iostream>

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

Node* createNode(int data) {
    Node* newNode = new Node();
    if (!newNode) {
        std::cout << "Gagal membuat node baru" << std::endl;
        exit(1);
    }
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

void addNodeDepan(Node*& head, int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
    }
    else {
        newNode->next = head;
        head = newNode;
    }
}

void addNodeBelakang(Node*& head, int data) {
    Node* newNode = createNode(data);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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

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

    Node* temp = head;
    while (temp->next != nullptr) {
        if (temp ->next ->data == data) {
            Node* nodeToDelete = temp->next;
            temp->next = temp->next->next;
            delete nodeToDelete;
            return;
        }
        temp = temp->next;
    }
}

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

int main() {
    Node* head = nullptr;

    addNodeDepan(head, 10);

    addNodeBelakang(head, 20);

    addNodeDepan(head, 5);

    std::cout << "Sebelum dihapus : ";
    printLinkedList(head);

    deleteNode(head, 10);

    std::cout << "Setelah setelah dihapus : ";
    printLinkedList(head);

    return 0;
}
```

b. Penjelasan Syntax

struct Node :Mendefinisikan struktur Node, yang mewakili elemen dalam linked list. Setiap Node memiliki dua bagian: int data untuk menyimpan nilai, dan Node* next yang menunjuk ke node berikutnya dalam linked list.

Node* createNode(int data) : Mencoba mengalokasikan memori untuk node baru. Jika alokasi gagal, menampilkan pesan kesalahan dan menghentikan program. Jika berhasil, mengisi data node dengan nilai yang diberikan, dan next diatur ke nullptr (karena node baru tidak terhubung ke node lain). Mengembalikan pointer ke node baru.

void addNodeDepan(Node*& head, int data) : Membuat node baru dengan createNode(data). Jika list kosong (head adalah nullptr), node baru menjadi head. Jika list tidak kosong, node baru dihubungkan ke head dan head diperbarui ke node baru, sehingga node baru berada di depan.

void addNodeBelakang(Node*& head, int data) : Membuat node baru

dengan createNode(data). Jika list kosong (head adalah nullptr), node baru menjadi head. Jika list tidak kosong, mencari node terakhir, lalu menambahkan node baru di akhir list dengan menghubungkannya ke node terakhir.

void deleteNode(Node*& head, int data) : Jika list kosong, keluar dari fungsi. Jika node pertama memiliki nilai yang sesuai, node pertama dihapus dan head diperbarui ke node berikutnya. Jika nilai berada di node lain, fungsi mencari node yang akan dihapus, lalu memperbarui pointer node sebelumnya untuk melewati node yang dihapus.

void printLinkedList(Node* head) : Memulai dari node pertama (head), mencetak nilai setiap node hingga node terakhir.
Setelah mencetak nilai terakhir, mencetak nullptr untuk menunjukkan akhir list.

int main() : Membuat linked list kosong dengan Node* head = nullptr. Menambahkan beberapa node ke linked list dengan addNodeDepan() dan addNodeBelakang(). Mencetak isi linked list sebelum menghapus node. Menghapus node dengan nilai tertentu menggunakan deleteNode(). Mencetak isi linked list setelah penghapusan node.

c. Output

```
Sebelum dihapus : 5 -> 10 -> 20 -> nullptr
Setelah setelah dihapus : 5 -> 20 -> nullptr
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\unguided\output>
```

3. Mencari dan Menghitung Panjang Linked List

a. Syntax

```cpp
#include <iostream>

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

class SinglyLinkedList {
private:
    Node* head;

public:
    SinglyLinkedList() {
        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;
        }
    }

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

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

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

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

int main() {
    SinglyLinkedList list;


    list.insertAtFront(10);
    list.insertAtBack(20);
    list.insertAtFront(5);

    int valueToSearch = 20;
    if (list.searchNode(valueToSearch)) {
        std::cout << "Node bernilai " << valueToSearch << " berhasil ditemukan." << std::endl;
    } else {
        std::cout << "Node bernilai " << valueToSearch << " tidak berhasil ditemukan." << std::endl;
    }

    int length = list.countLength();
    std::cout << "Panjang linked list: " << length << std::endl;

    return 0;
}
```

b. Penjelasan Syntax

Kode ini dimulai dengan mengimpor pustaka iostream, yang diperlukan untuk menggunakan fitur input dan output standar. Di dalamnya, terdapat struktur Node yang merepresentasikan elemen dalam linked list. Struktur ini memiliki dua anggota: data, yang menyimpan nilai integer, dan next, yang merupakan pointer yang menunjuk ke node berikutnya.

Selanjutnya, terdapat kelas SinglyLinkedList, yang berfungsi sebagai representasi dari linked list itu sendiri. Kelas ini memiliki satu anggota privat, yaitu pointer head, yang menunjuk ke node pertama dalam list. Konstruktor SinglyLinkedList() menginisialisasi head menjadi nullptr, menandakan bahwa list tersebut kosong.

Kelas ini memiliki beberapa metode untuk mengelola linked list. Metode insertAtFront(int value) digunakan untuk menambahkan node baru di depan list. Dalam metode ini, sebuah node baru diinisialisasi dengan nilai yang diberikan, dan pointer next-nya diatur untuk menunjuk ke node yang sebelumnya menjadi head. Kemudian, head diperbarui untuk menunjuk ke

node baru.

Metode insertAtBack(int value) berfungsi untuk menambahkan node baru di belakang list. Jika list kosong, head akan diatur untuk menunjuk ke node baru. Jika tidak, metode ini akan menelusuri list hingga menemukan node terakhir dan menghubungkan next-nya dengan node baru.

Metode searchNode(int value) memungkinkan pencarian node dalam list berdasarkan nilai tertentu. Jika node dengan nilai yang dicari ditemukan, metode ini akan mengembalikan true, sebaliknya akan mengembalikan false.

Untuk menghitung panjang linked list, digunakan metode countLength(). Metode ini mengiterasi melalui setiap node hingga mencapai akhir list, menghitung jumlah node yang ada.

Metode printList() digunakan untuk mencetak semua elemen dalam linked list. Metode ini menelusuri list dan mencetak nilai data dari setiap node, dengan format panah (->) di antara node untuk menunjukkan hubungan antara mereka.

Akhirnya, terdapat destruktor ~SingleLinkedList(), yang bertanggung jawab untuk membebaskan memori yang dialokasikan untuk node saat objek SingleLinkedList dihapus. Destruktor ini menelusuri setiap node dalam list dan menghapusnya untuk mencegah kebocoran memori.

Di dalam fungsi main(), objek dari kelas SingleLinkedList dibuat, dan beberapa node ditambahkan ke list. Fungsi pencarian digunakan untuk mencari node tertentu, dan hasilnya dicetak. Selain itu, panjang linked list juga dihitung dan ditampilkan.

c. Output

```
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\unguided\output>
e'
Node bernilai : 20 berhasi ditemukan.
Panjang linked list: 3
PS C:\Users\LENOVO\OneDrive - Telkom University\Documents\ALL Matkul\StrukturData\pertemuan4\unguided\output>
```

V. KESIMPULAN

Melakukan praktik implementasi single linked list ini membantu memahami konsep dasar struktur data yang dinamis dan bagaimana elemen-elemen diatur secara berurutan melalui pointer. Praktik ini mencakup penambahan elemen di depan dan belakang, penghapusan elemen, serta traversal atau penelusuran elemen untuk menampilkan isinya. Dengan latihan ini, kita dapat memahami penggunaan pointer dalam linked list serta operasi dasar seperti insert, delete, dan print, yang sangat penting dalam pengelolaan data yang fleksibel.