LAPORAN PRAKTIKUM
MODUL 4
SINGLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. 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.

B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini

2.1 Linked List dengan Pointer

Linked list merupakan salah satu bentuk struktur data yang terdiri dari serangkaian elemen data yang saling terhubung. Setiap elemen atau node dalam linked list menyimpan dua informasi utama: data dan pointer yang menunjuk ke node berikutnya. Penggunaan linked list memungkinkan penyimpanan data yang fleksibel, baik dalam jumlah maupun ukuran. Linked list dapat menyimpan berbagai jenis data, mulai dari data tunggal seperti integer hingga data majemuk seperti record yang mengandung beberapa tipe data (contohnya data mahasiswa yang berisi nama, NIM, dan alamat).

Implementasi linked list dalam bahasa pemrograman dapat dilakukan menggunakan array atau pointer, akan tetapi disini dalam penggunaan pointer lebih umum dan fleksibel. Hal ini disebabkan pointer dapat membuat linked list untuk bertambah dan berkurang sesuai kebutuhan tanpa batasan ukuran yang tetap seperti pada array. Selain itu, sifat pointer yang dinamis dan kemampuannya untuk menghubungkan elemen-elemen secara langsung membuat linked list lebih efisien dalam operasi penyisipan dan penghapusan dibandingkan array.

Pengaksesan elemen-elemen dalam linked list menggunakan pointer dapat dilakukan dengan operator -> atau . tergantung dari bahasa pemrograman yang digunakan. Beberapa model dari Abstract Data Type (ADT) linked list

yang sering dipelajari meliputi Single Linked List, Double Linked List, Circular Linked List, Stack, Queue, Tree, dan Graph. Setiap model ADT memiliki karakteristik unik yang disesuaikan dengan kebutuhan operasional tertentu.

Secara umum, operasi dasar yang dilakukan pada linked list meliputi penciptaan list, penyisipan elemen, penghapusan elemen, penelusuran elemen, pencarian elemen, serta pengubahan isi elemen.

2.2 Single Linked List

Single Linked List adalah jenis linked list yang hanya memiliki satu arah, di mana setiap node hanya menunjuk ke node berikutnya. Dalam single linked list, setiap elemen atau node terdiri dari dua bagian utama:

1. **Data**: Berisi informasi yang ingin disimpan di dalam linked list.
2. **Pointer**: Berfungsi sebagai penghubung antar elemen dengan menunjuk ke node berikutnya dalam list.

Ciri khas dari single linked list adalah:

- Hanya menggunakan satu pointer untuk menunjuk ke node berikutnya.
- Node terakhir di dalam linked list menunjuk ke NULL (kecuali pada circular list).
- Hanya bisa dilakukan maju, dari node pertama hingga node terakhir.
- Untuk pencarian elemen, jika data tidak terurut, maka pencarian harus dilakukan secara sekuensial.
- Penyisipan dan penghapusan elemen di tengah linked list relatif lebih mudah dibandingkan dengan array.

Istilah-istilah penting dalam single linked list termasuk:

- **First/Head**: Pointer yang menunjuk ke elemen pertama dalam linked list.
- **Next**: Pointer yang menunjuk ke node berikutnya.

- **Null/Nil**: Menunjukkan bahwa node tersebut tidak menunjuk ke mana pun, biasanya digunakan di node terakhir.
- **Node/Elemen**: Bagian dari linked list yang menyimpan data dan pointer ke elemen berikutnya.

2.2.1 Pembentukan Komponen-Komponen List

Operasi penelusuran atau view adalah proses menampilkan seluruh isi dari linked list. Operasi ini dilakukan dengan mencari elemen-elemen di dalam list, dimulai dari elemen pertama hingga elemen terakhir, dan menampilkan data yang tersimpan dalam setiap node.

Penelusuran ini dilakukan dengan cara mengunjungi setiap node secara berurutan, mulai dari node pertama yang ditunjuk oleh head, hingga mencapai node terakhir yang menunjuk ke NULL. Seluruh data dari setiap node ditampilkan selama traversal berlangsung.

2.2.2 Insert

Operasi penyisipan dalam linked list digunakan untuk menambahkan elemen baru ke dalam list. Ada beberapa jenis penyisipan yang umum dilakukan dalam single linked list:

- **Insert First**: Penyisipan elemen baru di awal linked list. Dalam operasi ini, node baru ditempatkan sebagai elemen pertama dalam list, dan pointer next dari node baru akan menunjuk ke elemen yang sebelumnya menjadi node pertama.
- **Insert Last**: Penyisipan elemen baru di akhir linked list. Untuk melakukan ini, kita harus menelusuri list dari elemen pertama hingga menemukan node terakhir, kemudian elemen baru ditambahkan setelah node terakhir tersebut.
- **Insert After**: Penyisipan elemen baru setelah node tertentu yang sudah ada di dalam list. Operasi ini memerlukan melewatihingga mencapai node yang diinginkan, dan kemudian node baru disisipkan di antara node yang ditunjuk dan node penerusnya.

2.2.3 View

Operasi penelusuran atau view adalah proses menampilkan seluruh isi dari linked list. Operasi ini dilakukan dengan menelusuri elemen-elemen di dalam list, dimulai dari elemen pertama hingga elemen terakhir, dan menampilkan data yang tersimpan dalam setiap node.

Penelusuran ini dilakukan dengan cara mengunjungi setiap node secara berurutan, mulai dari node pertama yang ditunjuk oleh head, hingga mencapai node terakhir yang menunjuk ke NULL. Seluruh data dari setiap node ditampilkan selama traversal berlangsung.

2.3 Delete

Operasi penghapusan pada linked list digunakan untuk menghapus elemen dari dalam list. Ada beberapa metode penghapusan yang dapat dilakukan pada single linked list:

- **Delete First**: Menghapus elemen pertama dari linked list. Operasi ini dilakukan dengan mengubah pointer head untuk menunjuk ke node kedua, dan kemudian node pertama dihapus dari memori.
- **Delete Last**: Menghapus elemen terakhir dalam linked list. Untuk melakukan ini, kita harus menelusuri list hingga mencapai node kedua terakhir, kemudian node terakhir dihapus, dan pointer next dari node kedua terakhir diset ke NULL.
- **Delete After**: Menghapus node yang terletak setelah node tertentu dalam list. Proses ini memerlukan penelusuran hingga node yang diinginkan, kemudian node berikutnya dihapus, dan pointer next dari node saat ini dihubungkan langsung dengan node setelah node yang dihapus.
- **Delete Element**: Menghapus elemen tertentu dari list, yang membutuhkan penelusuran untuk menemukan node dengan nilai yang sesuai, kemudian node tersebut dihapus dan memori yang dialokasikan dibebaskan.

2.4 Update

Operasi update digunakan untuk memperbarui data yang ada di dalam node tertentu dalam linked list. Operasi ini umumnya diawali dengan proses pencarian node yang ingin di-update berdasarkan nilai data yang disimpan di dalamnya. Setelah node yang diinginkan ditemukan, data di dalam node tersebut diperbarui dengan nilai yang baru.

Operasi update ini memungkinkan pengguna untuk mengganti nilai elemen tanpa harus menghapus node tersebut dari linked list. Proses update sering kali digunakan ketika data di dalam list mengalami perubahan tetapi strukturnya tetap sama

C. Guided

a. SLL Dengan Penambahan Dan Penghapusan di Depan dan Belakang

Code :

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

// // SLL dengan penambahan dan penghapusan elemen di depan dan belakang

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi List
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
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;
    }
}

// Tambah Belakang
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;
    }
}

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

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

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

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " << current-
>data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
    cout << endl; // Spasi antar tampilan
}

// Hapus List
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "123456"};
    mahasiswa m2 = {"Bob", "654321"};
    mahasiswa m3 = {"Charlie", "112233"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

Output :

```
PS D:\Kuliah\Struktur Data\Github\04_Single_Linked_List_Bagian_1\Guided>
pp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Nama: Alice, NIM: 123456

Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Charlie, NIM: 112233
Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Alice, NIM: 123456
Nama: Bob, NIM: 654321

Nama: Alice, NIM: 123456

List berhasil terhapus!
PS D:\Kuliah\Struktur Data\Github\04_Single_Linked_List_Bagian_1\Guided>
```

Penjelasan : Program C++ ini mengimplementasikan struktur data Single Linked List (SLL) untuk menyimpan informasi mahasiswa, yang terdiri dari nama dan NIM (Nomor Induk Mahasiswa). Terdapat dua struct utama: mahasiswa untuk menyimpan data mahasiswa dan Node untuk membentuk linked list dengan pointer yang mengarah ke node berikutnya. Variabel global head dan tail digunakan untuk menunjuk ke node pertama dan terakhir dalam list. Fungsi-fungsi utama dalam program ini mencakup init() untuk menginisialisasi list, isEmpty() untuk memeriksa apakah list kosong, insertDepan() dan insertBelakang() untuk menambahkan elemen di depan dan belakang list, serta hitungList() untuk menghitung jumlah elemen. Selain itu, terdapat juga hapusDepan() dan hapusBelakang() untuk menghapus elemen dari depan dan belakang list, dan tampil() untuk menampilkan semua elemen dalam list. Fungsi clearList() digunakan untuk menghapus semua elemen dan mengosongkan list. Di dalam fungsi main, list diinisialisasi, mahasiswa ditambahkan, dan list ditampilkan setelah setiap operasi.

b. SLL Dengan Penambahan di Awal dan Akhir serta Penghapusan Total

Code :

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

// Definisi struktur untuk elemen list
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        newNode->next = head;     // Menghubungkan elemen baru ke elemen pertama
        head = newNode;           // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}

// Menghitung jumlah elemen dalam list
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;              // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count;          // Mengembalikan jumlah elemen
}

// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;  // Membuat list kosong

    // Menambahkan elemen ke dalam list
    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

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

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

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

Output :

```
List berhasil terhapus!
PS D:\Kuliah\Struktur Data\Github\04_Single_Linked_List_Bagian_1\Guided>
d2 } ; if ($?) { .\guided2 }
Isi List: 10 20 30
Jumlah elemen: 3

Isi List setelah penghapusan: List kosong!
PS D:\Kuliah\Struktur Data\Github\04_Single_Linked_List_Bagian_1\Guided>
```

Penjelasan : Program C++ ini mengimplementasikan struktur data linked list sederhana untuk menyimpan elemen bertipe integer. Terdapat struct Node, yang mewakili elemen dalam linked list dengan dua komponen: data untuk menyimpan nilai elemen dan next, pointer ke elemen berikutnya. Fungsi utama yang diimplementasikan termasuk alokasi untuk mengalokasikan memori node baru; dealokasi untuk mengembalikan memori yang digunakan; dan isListEmpty untuk memeriksa apakah list kosong. Selain itu, ada fungsi insertFirst untuk menambah elemen di awal list dan insertLast untuk menambahkannya di akhir. Fungsi printList menampilkan semua elemen, sedangkan countElements menghitung jumlah elemen. Untuk menghapus semua elemen, terdapat fungsi clearList, yang mengembalikan memori setiap node. Dalam fungsi main, list diinisialisasi, dan elemen (10, 20, dan 30) ditambahkan. Setelah menampilkan isi list dan jumlah elemen, program menghapus seluruh elemen dan menampilkan kembali isi list untuk menunjukkan bahwa list kosong. Dengan demikian, program ini menggambarkan cara mengelola linked list, termasuk penambahan, pencetakan, penghitungan, dan penghapusan elemen.

D. Unguided

a. Membuat Single Linked List

Code:

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

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

Node* head = nullptr;

// Fungsi untuk menambah node di depan
void insertDepan(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

// Fungsi untuk mencetak linked list
void cetakList() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main() {
    insertDepan(10);
    insertBelakang(20);
    insertDepan(5);
    cetakList();

    return 0;
}
```

Output:

```
tSingleLinkedList.cpp -o Unguided_01_MembuatSingleLinkedList } ; if ($?) {
5 -> 10 -> 20 -> NULL
PS D:\Kuliah\Struktur Data\Github\04_Single_Linked_List_Bagian_1\Unguided>
0 ⚠ 0 ⓘ 198  0  Connect
```

Penjelasan:

Pada kode ini dibuat sebuah struktur data dasar untuk linked list, yaitu struct Node, yang memiliki dua elemen: data untuk menyimpan nilai integer, dan next yang merupakan pointer ke node berikutnya dalam list.

- **Deklarasi Node* head**: Pointer head digunakan untuk menunjuk ke node pertama (head) dari linked list. Saat linked list kosong, head akan berisi nullptr.
- **Fungsi insertDepan(int nilai)**: Fungsi ini menambahkan node baru di bagian depan linked list. Langkah-langkahnya:
    - Pertama, node baru (newNode) dibuat dengan new Node() dan yang kemudian nilainya diisi dengan parameter nilai.
    - Kemudian, newNode->next dihubungkan dengan node saat ini yang ditunjuk oleh head, sehingga node baru menjadi node pertama dan node sebelumnya menjadi node kedua.
    - Terakhir, head diperbarui menjadi newNode, sehingga pointer head sekarang menunjuk ke node yang baru saja ditambahkan di depan.
- **Fungsi insertBelakang(int nilai)**: Fungsi ini menambahkan node di akhir linked list. Langkah-langkahnya:
    - Sama seperti insertDepan(), node baru dibuat dengan nilai yang diberikan.
    - Jika linked list masih kosong (head == nullptr), maka node baru ini langsung menjadi head.
    - Jika tidak, maka proses akan dilakukan di dalam linked list dari head hingga menemukan node terakhir (node yang pointer next-nya adalah nullptr).
    - Setelah node terakhir ditemukan, maka akan dihubungkan node baru dengan node terakhir tersebut melalui temp->next = newNode.
- **Fungsi cetakList()**: Fungsi ini mencetak seluruh isi linked list. Langkah-

langkahnya:

- Dimulai dari head, kita melewati melalui setiap node, mencetak nilai data di setiap node, dan diikuti dengan tanda panah (->) yang menandakan hubungan antar node.
- Proses ini berakhir ketika node terakhir (yang next-nya adalah nullptr) ditemukan, dan program menampilkan "NULL" sebagai penutup, menandakan akhir dari linked list.

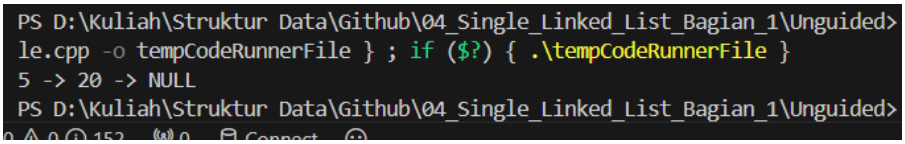
Contoh hasil output dari program ini adalah urutan nilai node dari yang pertama hingga terakhir, misalnya "8 -> 15 -> 25 -> NULL" setelah beberapa operasi penambahan node.

b. Menghapus Node pada Linked List

Code:

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

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

Node* head = nullptr;

// Fungsi untuk menambah node di depan
void insertDepan(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

// Fungsi untuk menghapus node berdasarkan nilai tertentu
void hapusNode(int nilai) {
    Node* temp = head;
    Node* prev = nullptr;

    if (temp != nullptr && temp->data == nilai) {
        head = temp->next;
        delete temp;
        return;
    }

    while (temp != nullptr && temp->data != nilai) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr) return;

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

// Fungsi untuk mencetak linked list
void cetakList() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " -> ";
        temp = temp->next;
    }
    cout << "NULL" << endl;
}

int main() {
    insertDepan(10);
    insertBelakang(20);
    insertDepan(5);
    hapusNode(10);
    cetakList();

    return 0;
}
```

Output:



Penjelasan:

Pada kode ini memiliki fungsi yang memungkinkan untuk menghapus node tertentu berdasarkan nilai yang diberikan.

- **Fungsi hapusNode(int nilai)**: Fungsi ini bertugas untuk menghapus node dengan nilai tertentu. Langkah-langkahnya:
    - Pertama, diperiksa apakah head mengandung nilai yang akan dihapus. Jika iya, head langsung diubah ke node berikutnya (head = temp->next) dan node lama yang berisi nilai dihapus dari memori (delete temp).
    - Jika nilai yang akan dihapus tidak ada di head, maka dilakukan proses lanjutan pada linked list untuk menemukan node yang memiliki nilai yang sesuai dengan apa yang dicari.
    - Dalam proses ini, dua pointer digunakan: temp untuk menunjuk node yang sedang diperiksa, dan prev untuk menunjuk node sebelum temp.
    - Ketika node dengan nilai yang cocok ditemukan, node tersebut dihapus dengan cara mengubah prev->next sehingga langsung menunjuk ke node setelah temp, kemudian node temp dihapus.
    - Jika node yang dicari tidak ditemukan (temp == nullptr), fungsi selesai tanpa melakukan apapun.
- **Fungsi cetakList()** setelah penghapusan node mencetak isi linked list yang sudah diperbarui tanpa node yang dihapus.

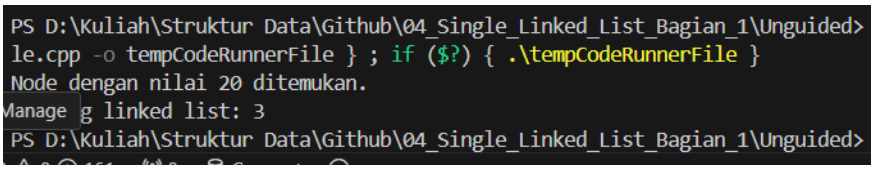
Contoh outputnya, setelah node dengan nilai 15 dihapus, linked list akan menampilkan "8 -> 25 -> NULL", menandakan bahwa node 15 berhasil dihapus.

c. Mencari dan Menghitung Panjang Linked List

Code:

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

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

Node* head = nullptr;

// Fungsi untuk menambah node di depan
void insertDepan(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = head;
    head = newNode;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(int nilai) {
    Node* newNode = new Node();
    newNode->data = nilai;
    newNode->next = nullptr;

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

// Fungsi untuk mencari node dengan nilai tertentu
bool cariNode(int nilai) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == nilai) return true;
        temp = temp->next;
    }
    return false;
}

// Fungsi untuk menghitung panjang linked list
int hitungPanjang() {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

int main() {
    insertDepan(10);
    insertBelakang(20);
    insertDepan(5);

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

    cout << "Panjang linked list: " << hitungPanjang() << endl;

    return 0;
}
```

Output:



Penjelasan:

Pada kode ini terdapat dua fungsi: satu untuk mencari node berdasarkan nilai, dan satu lagi untuk menghitung jumlah node di linked list.

- **Fungsi cariNode(int nilai)**: Fungsi ini digunakan untuk mencari apakah ada node dengan nilai tertentu di dalam linked list. Langkah-langkahnya:
    - Fungsi ini mengecek setiap node mulai dari head sampai tail atau akhir dan memeriksa apakah data di node tersebut sama dengan nilai yang dicari.
    - Jika nilai yang dicari ditemukan, fungsi mengembalikan nilai “true” dan pencarian berhenti.
    - Jika pencarian sampai ke akhir linked list tanpa menemukan nilai yang cocok, fungsi mengembalikan “false”, dari hal tersebut maka diketahui bahwa nilai tersebut tidak ada di dalam linked list.
- **Fungsi hitungPanjang()**: Fungsi ini berfungsi untuk menghitung jumlah node di dalam linked list. Langkah-langkahnya:
    - Dimulai dari node pertama (head), fungsi ini mengecek setiap node dan menambah variabel penghitung (count) sampai pada node terakhir.
    - Setelah mencapai akhir (node dengan next == nullptr), fungsi mengembalikan nilai dari count, yang menunjukkan berapa banyak node yang ada di linked list.

Setelah menjalankan program, jika pengguna menambahkan node dengan nilai 8, 15, dan 25, dan kemudian mencari node dengan nilai 25, program akan menampilkan bahwa node tersebut ditemukan. Selain itu, jika pengguna menghitung panjang linked list, program akan mengeluarkan angka 3, yang menunjukkan bahwa ada 3 node di dalam linked list.

E. Kesimpulan

Pada praktikum ini, dipelajari implementasi struktur data Single Linked List (SLL). SLL merupakan struktur data dinamis yang tersusun dari elemen-elemen yang saling terhubung melalui pointer, memungkinkan penyimpanan data secara fleksibel. Beberapa operasi dasar yang diimplementasikan meliputi penyisipan elemen di awal atau akhir list (insert), penghapusan elemen di berbagai posisi (delete), penelusuran seluruh elemen (view), pencarian elemen dengan nilai tertentu (search), serta penghitungan jumlah elemen dalam list (count). Operasi-operasi ini menunjukkan bagaimana data dapat dikelola secara efisien menggunakan SLL.

SLL memiliki keunggulan dalam efisiensi penyisipan dan penghapusan dibandingkan array, karena tidak memerlukan pergeseran elemen lain saat ada perubahan data. Namun, pencarian elemen di SLL harus dilakukan secara sekuensial dari awal hingga akhir, yang menjadi kelemahan jika dibandingkan dengan array yang memungkinkan akses langsung ke elemen tertentu. Meskipun demikian, praktikum ini memberikan pemahaman yang mendalam tentang pengelolaan memori dinamis serta operasi dasar pada linked list, yang sangat penting dalam pengembangan aplikasi dengan data yang berubah-ubah.