LAPORAN PRAKTIKUM

Modul 04

“SINGLE LINKED LIST (BAGIAN PERTAMA)”

Disusun Oleh:

Marvel Sanjaya Setiawan (2311104053)

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. **Tujuan**
    - Membangun program yang memanfaatkan struktur data linked list.
    - Belajar memprogram dengan linked list.
    - Mempelajari dan menerapkan linked list dalam pemrograman.
2. **Landasan Teori**

    Linked List

    Linked List adalah struktur data yang terdiri dari node-node yang saling terhubung melalui pointer. Setiap node berisi data dan pointer ke node berikutnya.

    Kelebihan Linked List:

    - Fleksibel: Mudah menambah atau menghapus elemen.
    - Dinamis: Ukurannya dapat berubah sesuai kebutuhan.

    Struktur Node:

    - Data: Menyimpan informasi yang ingin disimpan.
    - Pointer: Menunjuk ke node berikutnya.

    Jenis-jenis Linked List:

    - Single Linked List: Setiap node hanya memiliki satu pointer ke node berikutnya.
    - Double Linked List: Setiap node memiliki dua pointer, ke node sebelumnya dan berikutnya.
    - Circular Linked List: Node terakhir menunjuk ke node pertama.

Operasi Dasar:

- Create: Membuat list baru.
- Insert: Menambahkan node ke list.
- Delete: Menghapus node dari list.
- Search: Mencari node tertentu.
- Traverse: Melalui semua node dalam list.

Implementasi:

- Menggunakan pointer untuk menghubungkan node-node.
- Alokasi memori dinamis untuk setiap node baru.

Contoh Penggunaan:

- Menyimpan data yang ukurannya tidak tetap.
- Mengimplementasikan struktur data lain seperti stack, queue, tree, dan graph.

.

Telkom University Purwokerto

3. Guided

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

// Deklarasi Struct untuk mahasiswa struct 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()
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;
}

// 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;
}
```

```
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!
```

Cara kerja program:

1. Deklarasi struktur: Program mendefinisikan struktur mahasiswa untuk menyimpan data mahasiswa dan struktur Node untuk mewakili setiap node dalam linked list.
2. Inisialisasi: Fungsi init() membuat linked list kosong.
3. Operasi: Program melakukan berbagai operasi pada linked list, seperti menambahkan, menghapus, dan menampilkan data.
4. Contoh penggunaan: Program memberikan contoh cara menggunakan fungsi-fungsi yang telah didefinisikan.

Telkom University Purwokerto

```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
    }
}

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;
}
```

```
Isi List: 10 20 30
Jumlah elemen: 3
Isi List setelah penghapusan: List kosong!
```

Cara kerja:

1. Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2. Operasi: Program menyediakan berbagai operasi pada linked list, seperti menambahkan, menghapus, dan menampilkan elemen.
3. Alokasi dan dealokasi memori: Fungsi alokasi dan dealokasi digunakan untuk mengelola memori secara efisien.
4. Pointer head: Pointer head menunjuk ke node pertama dalam list.

4. Unguided

1. Membuat Single Linked List.

```cpp
//Membuat SLL
#include <iostream>
using namespace std;

// Definisi struktur untuk elemen linked list
struct Node {
    int data;
    Node* next;
};

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

// Fungsi untuk menambah node di belakang
void insertBelakang(Node* &head, 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;
    }
}

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

// Main function untuk Soal 1
int main() {
    Node* head = nullptr;

    // Menambahkan node ke linked list
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

    // Mencetak isi linked list
    cout << "Isi Linked List: ";
    cetakList(head);

    return 0;
}
```

```
Isi Linked List: 5 -> 10 -> 20
```

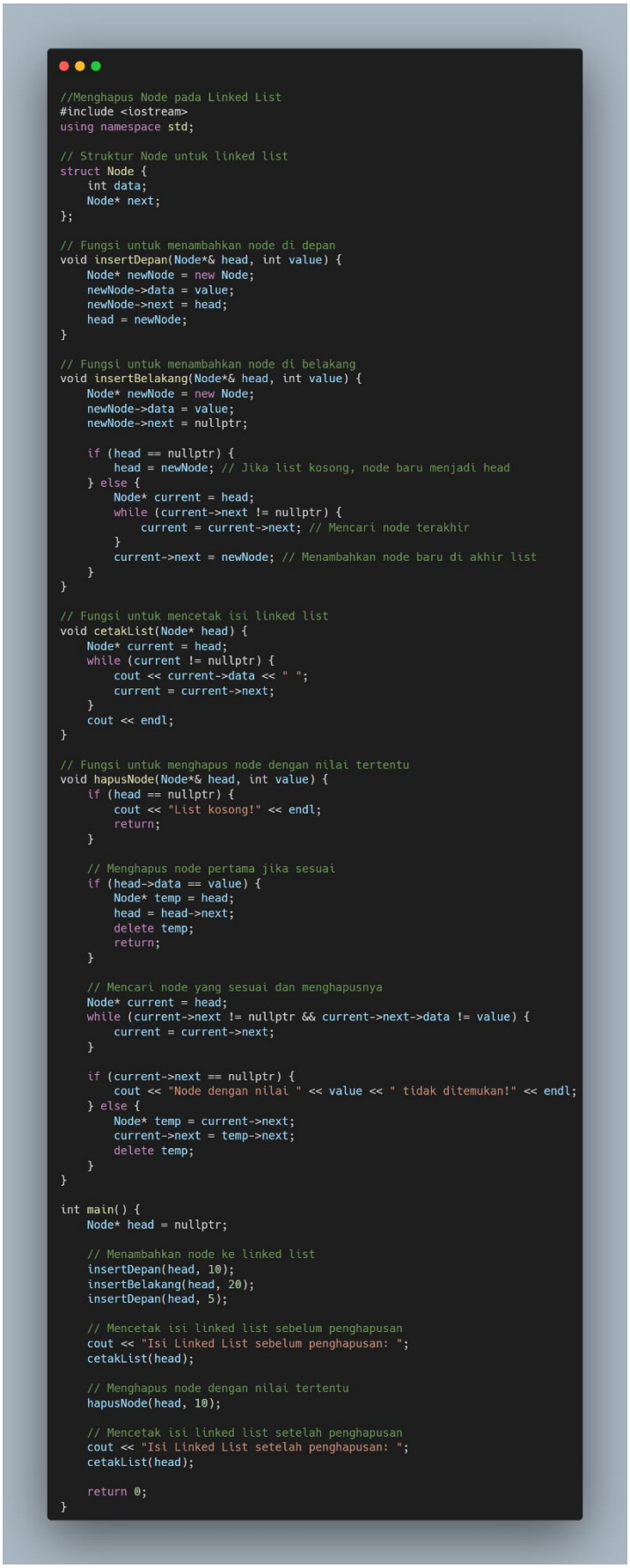
Fungsi-fungsi utama:

- insertDepan: Menambahkan node di awal linked list.
- insertBelakang: Menambahkan node di akhir linked list.
- cetakList: Mencetak semua elemen dalam linked list.

Cara kerja:

1) Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2) Operasi: Program menyediakan fungsi untuk menambahkan node di awal dan akhir linked list.
3) Pointer head: Pointer head menunjuk ke node pertama dalam linked list.

2. Menghapus Node pada Linked List



```cpp
//Menghapus Node pada Linked List
#include <iostream>
using namespace std;

// Struktur Node untuk linked list
struct Node {
    int data;
    Node* next;
};

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

// Fungsi untuk menambahkan node di belakang
void insertBelakang(Node*& head, int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode; // Jika list kosong, node baru menjadi head
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next; // Mencari node terakhir
        }
        current->next = newNode; // Menambahkan node baru di akhir list
    }
}

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

// Fungsi untuk menghapus node dengan nilai tertentu
void hapusNode(Node*& head, int value) {
    if (head == nullptr) {
        cout << "List kosong!" << endl;
        return;
    }

    // Menghapus node pertama jika sesuai
    if (head->data == value) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    // Mencari node yang sesuai dan menghapusnya
    Node* current = head;
    while (current->next != nullptr && current->next->data != value) {
        current = current->next;
    }

    if (current->next == nullptr) {
        cout << "Node dengan nilai " << value << " tidak ditemukan!" << endl;
    } else {
        Node* temp = current->next;
        current->next = temp->next;
        delete temp;
    }
}

int main() {
    Node* head = nullptr;

    // Menambahkan node ke linked list
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

    // Mencetak isi linked list sebelum penghapusan
    cout << "Isi Linked List sebelum penghapusan: ";
    cetakList(head);

    // Menghapus node dengan nilai tertentu
    hapusNode(head, 10);

    // Mencetak isi linked list setelah penghapusan
    cout << "Isi Linked List setelah penghapusan: ";
    cetakList(head);

    return 0;
}
```

```
Isi Linked List sebelum penghapusan: 5 10 20
Isi Linked List setelah penghapusan: 5 20
```

Fungsi-fungsi utama:

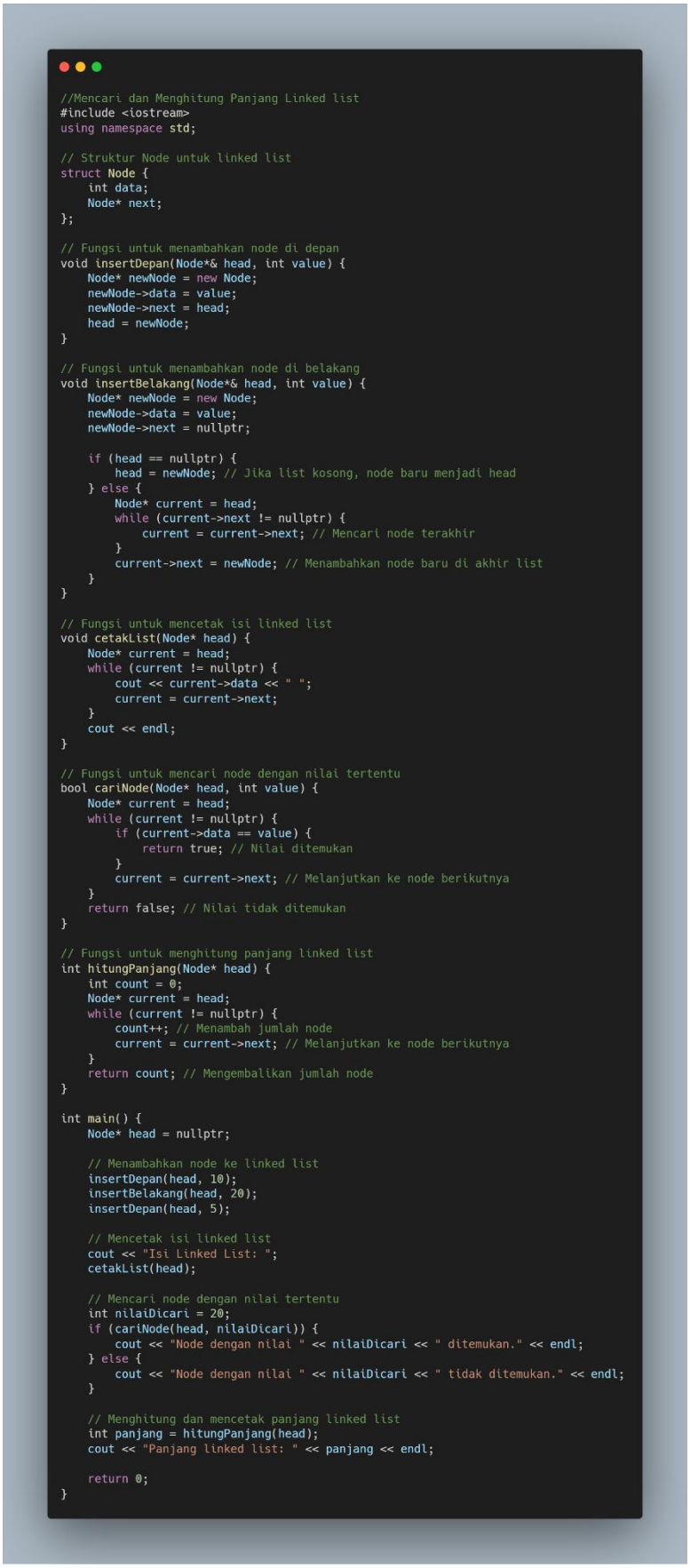
- insertDepan: Menambahkan node di awal linked list.
- insertBelakang: Menambahkan node di akhir linked list.
- hapusNode: Menghapus node dengan nilai tertentu dari linked list.
- cetakList: Mencetak semua elemen dalam linked list.

Cara Kerja:

1) Struktur Node: Setiap node memiliki dua anggota: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya.
2) Operasi: Program menyediakan fungsi untuk menambahkan, menghapus, dan menampilkan elemen dalam linked list.
3) Pointer head: Pointer head menunjuk ke node pertama dalam linked list.

Telkom University Purwokerto

3. Mencari dan Menghitung Panjang Linked List



```cpp
//Mencari dan Menghitung Panjang Linked list
#include <iostream>
using namespace std;

// Struktur Node untuk linked list
struct Node {
    int data;
    Node* next;
};

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

// Fungsi untuk menambahkan node di belakang
void insertBelakang(Node*& head, int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        head = newNode; // Jika list kosong, node baru menjadi head
    } else {
        Node* current = head;
        while (current->next != nullptr) {
            current = current->next; // Mencari node terakhir
        }
        current->next = newNode; // Menambahkan node baru di akhir list
    }
}

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

// Fungsi untuk mencari node dengan nilai tertentu
bool cariNode(Node* head, int value) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == value) {
            return true; // Nilai ditemukan
        }
        current = current->next; // Melanjutkan ke node berikutnya
    }
    return false; // Nilai tidak ditemukan
}

// Fungsi untuk menghitung panjang linked list
int hitungPanjang(Node* head) {
    int count = 0;
    Node* current = head;
    while (current != nullptr) {
        count++; // Menambah jumlah node
        current = current->next; // Melanjutkan ke node berikutnya
    }
    return count; // Mengembalikan jumlah node
}

int main() {
    Node* head = nullptr;

    // Menambahkan node ke linked list
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

    // Mencetak isi linked list
    cout << "Isi Linked List: ";
    cetakList(head);

    // Mencari node dengan nilai tertentu
    int nilaiDicari = 20;
    if (cariNode(head, nilaiDicari)) {
        cout << "Node dengan nilai " << nilaiDicari << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << nilaiDicari << " tidak ditemukan." << endl;
    }

    // Menghitung dan mencetak panjang linked list
    int panjang = hitungPanjang(head);
    cout << "Panjang linked list: " << panjang << endl;

    return 0;
}
```

```
Isi Linked List: 5 10 20
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

Fungsi-fungsi dasar:

- Menambahkan node: Fungsi insertDepan dan insertBelakang untuk menambahkan node di awal dan akhir list.
- Mencetak list: Fungsi cetakList untuk menampilkan semua data dalam list.
- Mencari node: Fungsi cariNode untuk mencari keberadaan suatu nilai dalam list.
- Menghitung panjang: Fungsi hitungPanjang untuk menghitung jumlah node dalam list.

Cara Kerja:

1) Struktur Node: Setiap node memiliki data dan pointer ke node berikutnya.
2) Pointer head: Menunjuk ke node pertama dalam list.
3) Operasi: Fungsi-fungsi di atas melakukan operasi pada linked list, seperti menambahkan, mencari, dan menghitung node.

5. Kesimpulan

Laporan praktikum ini telah berhasil mengimplementasikan struktur data linked list secara sederhana. Melalui berbagai fungsi yang telah diimplementasikan, seperti penambahan, penghapusan, pencarian, dan penghitungan node, dapat disimpulkan bahwa linked list merupakan struktur data yang fleksibel dan dinamis.