I. COVER

LAPORAN PRAKTIKUM

Modul 4

SINGLE LINKED LIST

Disusun Oleh:

Haza Zaidan Zidna Fann

(2311104056)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

II. TUJUAN

Memahami penggunaan *linked list* dengan *pointer* operator- operator dalam program.
Memahami operasi-operasi dasar dalam *linked list*.

Membuat program dengan menggunakan *linked list* dengan *prototype* yang ada

III. Landasan Teori

Linked List

Linked list adalah salah satu bentuk struktur data dinamis yang terdiri dari serangkaian elemen data yang saling berkaitan satu sama lain. Setiap elemen dalam linked list disebut sebagai "node", yang terdiri dari dua komponen utama:

Data: Merupakan informasi yang disimpan dalam node.

Pointer/Suksesor: Merujuk ke elemen berikutnya dalam list.

Jenis Linked List :

Single Linked List: Hanya memiliki satu arah pointer, sehingga data hanya dapat diakses sekuensial dari awal hingga akhir.

Double Linked List: Memiliki dua pointer, satu mengarah ke elemen sebelumnya dan satu lagi ke elemen berikutnya.

Circular Linked List: Pointer dari elemen terakhir menunjuk kembali ke elemen pertama.

Multi Linked List, Stack, Queue, Tree, dan Graph juga dapat direpresentasikan menggunakan konsep linked list.

Keunggulan Linked List :

Dinamis: Ukurannya bisa berubah sesuai kebutuhan.

Mudah dalam Penyisipan dan Penghapusan: Operasi penambahan atau penghapusan elemen tidak memerlukan pemindahan elemen-elemen lainnya.

Memori Lebih Efisien: Hanya memerlukan memori sesuai jumlah elemen.

Operasi Dasar Linked List:

Penciptaan dan Inisialisasi (Create List): Mengatur pointer awal dan akhir list ke nilai Nil.

Penyisipan (Insert): Memasukkan elemen baru pada awal, akhir, atau setelah elemen tertentu.

Penghapusan (Delete): Mengambil elemen dari awal, akhir, atau lokasi tertentu.

Penelusuran (View): Mengunjungi setiap node dan menampilkan informasi.

Pencarian (Searching): Menemukan elemen tertentu dalam list.

Pengubahan (Update): Mengganti nilai elemen.

IV. GUIDED

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

// 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;
    }
}

// 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!
```

Pendahuluan

implementasi dari struktur data *Single Linked List* untuk menyimpan data mahasiswa, yang terdiri dari nama dan NIM. Operasi yang dilakukan meliputi inisialisasi list, pengecekan apakah list kosong, penambahan dan penghapusan elemen, serta penampilan data.

2. **Deklarasi Struct**

mahasiswa: Menyimpan data berupa nama dan NIM.

Node: Mewakili elemen dalam linked list yang berisi data mahasiswa dan pointer next untuk menunjuk ke elemen berikutnya.

3. **Variabel Global**

head dan tail: Menunjuk ke elemen pertama dan terakhir dalam linked list.

4. **Fungsi-Fungsi Utama**

init(): Menginisialisasi list dengan mengatur head dan tail menjadi nullptr.

isEmpty(): Mengecek apakah list kosong dengan memeriksa apakah head adalah nullptr.

insertDepan(): Menambahkan elemen baru di depan list. Jika list kosong, head dan tail menunjuk ke elemen baru. Jika tidak, elemen baru ditempatkan di depan head.

insertBelakang(): Menambahkan elemen di belakang list. Jika list kosong, elemen baru menjadi head dan tail. Jika tidak, elemen baru ditempatkan di belakang tail.

hitungList(): Menghitung jumlah elemen dalam list dengan cara menelusuri setiap node.

hapusDepan(): Menghapus elemen di depan. Jika list hanya berisi satu elemen, maka head dan tail diatur menjadi nullptr.

hapusBelakang(): Menghapus elemen di belakang. Jika list hanya berisi satu elemen, maka head dan tail diatur menjadi nullptr. Jika tidak, elemen sebelum tail dijadikan sebagai elemen terakhir.

tampil(): Menampilkan semua elemen dalam list.

clearList(): Menghapus seluruh elemen dalam list dan mengatur head dan tail menjadi nullptr.

5. **Main Function**

Inisialisasi list.

Menambahkan data mahasiswa ke dalam list dan menampilkan hasil.

Menghapus elemen dari list dan menampilkan perubahan.

Menghapus seluruh list sebelum program selesai.

Alur Program :

Inisialisasi List (init())

Program memulai dengan memanggil fungsi init() untuk mengatur *head* dan *tail* menjadi nullptr, menandakan bahwa linked list masih kosong.

Menambahkan Data Mahasiswa

Program mendeklarasikan tiga data mahasiswa (m1, m2, m3).

Memasukkan data ke dalam linked list:

insertDepan(m1): Menambahkan mahasiswa m1 di depan list. Karena list kosong, m1 menjadi elemen pertama (head) dan terakhir (tail).

insertBelakang(m2): Menambahkan mahasiswa m2 di belakang. Pointer next dari tail (yang saat ini menunjuk ke m1) diarahkan ke elemen baru (m2), dan tail diperbarui menjadi m2.

insertDepan(m3): Menambahkan mahasiswa m3 di depan. m3 menjadi head baru dan menunjuk ke elemen sebelumnya (m1).

Menampilkan Isi List (tampil())

Program menampilkan isi linked list setelah setiap operasi penambahan, untuk memperlihatkan keadaan terkini list.

Menghapus Elemen

hapusDepan(): Menghapus elemen pertama (m3). head diperbarui menjadi elemen berikutnya (m1).

hapusBelakang(): Menghapus elemen terakhir (m2). tail diperbarui menjadi elemen sebelumnya (m1).

Menghapus Semua Elemen (clearList())

Menghapus semua elemen yang tersisa dan mengatur head dan tail menjadi nullptr, menandakan bahwa list telah dikosongkan.

Program Berakhir

Program selesai dan mengembalikan nilai 0.

```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!
```

1. **Node Struct**

   Node adalah struktur yang merepresentasikan elemen dalam linked list. Setiap elemen (Node) memiliki:

   data: Menyimpan nilai elemen.

   next: Pointer yang menunjuk ke elemen berikutnya.

2. **Fungsi Alokasi dan Dealokasi**

   Node* alokasi(int value): Mengalokasikan memori untuk Node baru dengan nilai value dan mengembalikan pointer ke Node tersebut.

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

3. **Fungsi Dasar List**

   bool isListEmpty(Node* head): Mengecek apakah list kosong dengan memeriksa apakah head adalah nullptr.

   void insertFirst(Node* &head, int value): Menambahkan elemen baru di awal list. Jika list kosong, elemen baru menjadi head.

   void insertLast(Node* &head, int value): Menambahkan elemen baru di akhir list. Jika list kosong, elemen baru menjadi head; jika tidak, elemen baru ditempatkan di akhir.

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

   int countElements(Node* head): Menghitung jumlah elemen dalam list dengan menelusuri dari head hingga akhir.

   void clearList(Node* &head): Menghapus semua elemen dalam list dan mengosongkan memori.

   Alur Program :

   Inisialisasi

   Membuat list kosong dengan Node* head = nullptr.

   Menambahkan Elemen

   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 dan Menghitung Elemen

printList(head): Menampilkan elemen yang ada di dalam list.

countElements(head): Menghitung dan menampilkan jumlah elemen dalam list.

Menghapus Seluruh Elemen

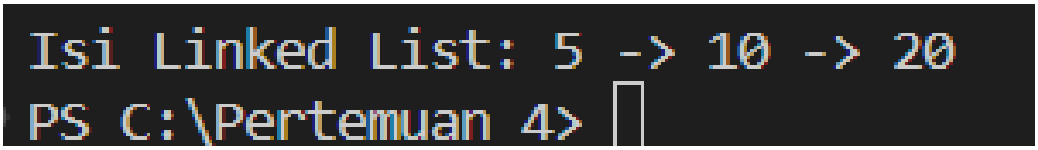
clearList(head): Menghapus semua elemen dari list dan membebaskan memori.

Mengecek Isi List Setelah Penghapusan

Menampilkan isi list setelah proses penghapusan untuk memastikan list kosong.

V. UNGUIDED

```cpp
#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;
}
```



1. **insertDepan(Node* &head, int value):**

   Menambahkan node baru di awal list.

   Menciptakan node baru dengan nilai yang diberikan, menghubungkannya ke node sebelumnya.

2. **insertBelakang(Node* &head, int value):**

   Menambahkan node baru di akhir list.

   Jika list kosong, node baru menjadi head. Jika tidak, traverse (telusuri) hingga node terakhir dan hubungkan node baru.

3. **cetakList(Node* head):**

   Mencetak seluruh isi linked list.

   Menggunakan loop untuk mengakses setiap node dan mencetak nilai hingga mencapai akhir list (nullptr).

   Alur Program :

   Di dalam fungsi main():

   Membuat list kosong dengan head diatur menjadi nullptr.

   Menambahkan beberapa node:

   insertDepan(head, 10): Menambahkan 10 di depan.

   insertBelakang(head, 20): Menambahkan 20 di belakang.

   insertDepan(head, 5): Menambahkan 5 di depan.

   Mencetak isi linked list dengan memanggil cetakList(head).

```cpp
#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
```

1. **insertDepan(Node*& head, int value):**

   Menambahkan node baru di awal list.

   Membuat node baru, mengisi data, dan mengatur next ke head saat ini.

2. **insertBelakang(Node*& head, int value):**

   Menambahkan node baru di akhir list.

   Jika list kosong, node baru menjadi head. Jika tidak, traverse hingga node terakhir dan menghubungkan node baru.

3. **cetakList(Node* head):**

   Mencetak semua elemen dalam list.

   Menggunakan loop untuk menelusuri dan mencetak data hingga mencapai akhir list (nullptr).

4. **hapusNode(Node*& head, int value):**

   Menghapus node yang memiliki nilai tertentu.

   Memeriksa jika list kosong; jika tidak, mencari node yang cocok dan menghapusnya. Jika node yang dicari tidak ditemukan, tampilkan pesan.

   Alur Program :

   Di dalam main():

   Membuat list kosong dengan head diatur menjadi nullptr.

   Menambahkan node ke dalam list:

   insertDepan(head, 10): Menambahkan 10 di depan.

   insertBelakang(head, 20): Menambahkan 20 di belakang.

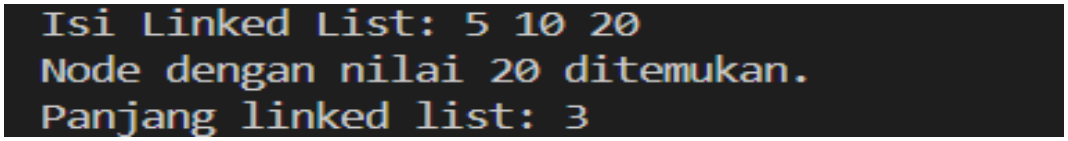
   insertDepan(head, 5): Menambahkan 5 di depan.

   Mencetak isi linked list sebelum penghapusan.

   Menghapus node dengan nilai 10 menggunakan hapusNode(head, 10).

   Mencetak isi linked list setelah penghapusan.

```cpp
#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;
}
```



1. **insertDepan(Node*& head, int value)**:

   Menambahkan node baru di awal list.

   Membuat node baru, mengisi data, dan mengatur next ke node yang ada sebelumnya.

2. **insertBelakang(Node*& head, int value)**:

   Menambahkan node baru di akhir list.

   Jika list kosong, node baru menjadi head. Jika tidak, traverse hingga node terakhir dan menghubungkan node baru.

3. **cetakList(Node* head)**:

   Mencetak semua elemen dalam linked list dengan loop hingga mencapai akhir list (nullptr).

4. **cariNode(Node* head, int value)**:

   Mencari apakah sebuah nilai ada dalam linked list.

   Mengembalikan true jika ditemukan, dan false jika tidak.

5. **hitungPanjang(Node* head)**:

   Menghitung jumlah node dalam linked list dengan loop dan mengembalikan hasilnya.

   Alur Program :

   Di dalam fungsi main():

   Membuat list kosong (head diatur ke nullptr).

   Menambahkan node dengan:

   insertDepan(head, 10): Menambahkan 10 di depan.

   insertBelakang(head, 20): Menambahkan 20 di belakang.

   insertDepan(head, 5): Menambahkan 5 di depan.

   Mencetak isi linked list dengan memanggil cetakList(head).

   Mencari node dengan nilai 20 menggunakan cariNode(head, 20), dan menampilkan hasil pencarian.

   Menghitung dan mencetak panjang linked list.

VI. Kesimpulan

Linked list adalah struktur data yang terdiri dari *node* yang menyimpan data dan pointer ke node berikutnya. Operasi dasar meliputi penambahan node, pencetakan elemen, pencarian nilai, dan penghitungan jumlah node. Keunggulannya dibandingkan array adalah fleksibilitas ukuran serta kemudahan dalam penyisipan dan penghapusan. Linked list sering digunakan dalam pengelolaan memori dan penyimpanan data dinamis.