LAPORAN PRAKTIKUM

MODUL

SINGLE LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

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

b. Memahami operasi-operasi dasar dalam *linked list.*

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

II. LANDASAN TEORI

1. *Linked List dengan Pointer*

linked list adalah salah satu struktur data yang terdiri dari serangkaian elemen yang saling terhubung, di mana setiap elemen memiliki data dan *pointer* yang menunjuk ke elemen berikutnya. Struktur ini fleksibel karena ukurannya dapat bertambah atau berkurang sesuai kebutuhan tanpa perlu mendefinisikan ukuran awal seperti pada array. Dalam implementasinya, *single linked list* hanya memerlukan satu arah *pointer*, sehingga setiap node hanya memiliki informasi untuk menghubungkan ke node berikutnya dan node terakhir menunjuk ke 'NULL' sebagai tanda akhir dari list.

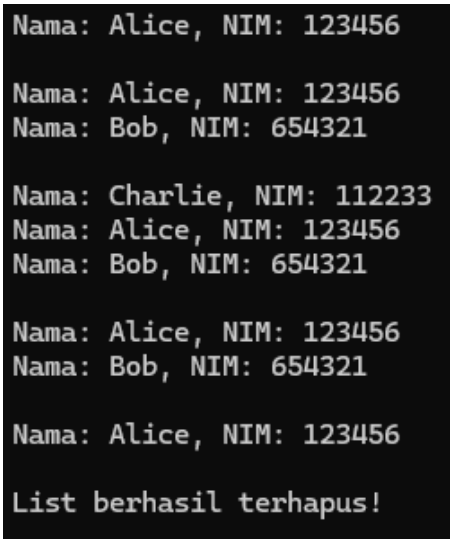
Operasi dasar yang dapat dilakukan pada single linked list mencakup pembuatan list (*create*), penambahan elemen di awal (*insert first*) dan di akhir (*insert last*), penghapusan elemen (*delete*), serta penelusuran elemen (*traverse*). Penggunaan pointer memungkinkan *linked list* untuk menyimpan elemen-elemen secara dinamis di memori. Hal ini mempermudah penambahan atau penghapusan elemen di posisi tertentu dibandingkan array yang bersifat statis. Selain itu, *linked list* juga mendukung pencarian *sequntial* untuk menemukan elemen berdasarkan nilai tertentu.

Dalam program yang melibatkan *linked list*, fungsi 'cariNode' digunakan untuk mencari apakah suatu nilai ada dalam list, sedangkan 'hitungPanjang' bertujuan untuk menghitung jumlah node atau panjang dari list. Keunggulan dari *single linked list* adalah kemampuannya untuk mengelola memori secara efisien sesuai dengan kebutuhan data yang ada, menjadikannya solusi yang efektif untuk berbagai masalah yang memerlukan struktur data yang dapat berubah-ubah ukurannya.

III. GUIDED

1. guided_1

```cpp
// 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;
        }
        cout << "\n";
    } else {
        cout << "List masih kosong!\n\n";
    }
}

// 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;
```



Program ini bertujuan untuk mengelola data mahasiswa dengan menggunakan struktur data single linked list. Single linked list adalah struktur data yang terdiri dari serangkaian elemen yang saling terhubung, di mana setiap elemen (node) berisi data dan pointer yang menunjuk ke node berikutnya. Dalam konteks ini, data yang disimpan adalah informasi mahasiswa, yaitu nama dan NIM.

a. **Struktur Data**
   - Struct ‘mahasiswa’: Digunakan untuk mendefinisikan data mahasiswa. Struct ini memiliki dua anggota:
     - ‘Nama’: Sebuah array karakter dengan panjang maksimal 30, digunakan untuk menyimpan nama mahasiswa.
     - ‘nim’: Sebuah array karakter dengan panjang maksimal 10, digunakan untuk menyimpan NIM mahasiswa.
   - Struct ‘Node’: Digunakan untuk mendefinisikan node dalam linked list. Setiap node memiliki dua anggota:
     - ‘data’: Menyimpan informasi mahasiswa dengan tipe struct ‘mahasiswa’.
     - ‘next’: Sebuah pointer yang menunjuk ke node berikutnya dalam list.

b. Variabel Global
   - ‘Node *head’: Pointer yang menunjuk ke node pertama dalam linked list.
   - ‘Node *tail’: Pointer yang menunjuk ke node terakhir dalam linked list.

c. Fungsi Inisialisasi
   - ‘void init()’: Fungsi ini digunakan untuk menginisialisasi linked list. Ia mengatur ‘head’ dan ‘tail’ menjadi ‘nullptr’, yang berarti list kosong.

d. Pengecekan Kosong
   - ‘bool isEmpty()’: Fungsi ini mengembalikan nilai boolean. Jika ‘head’ adalah ‘nullptr’, maka linked list dianggap kosong dan fungsi ini mengembalikan ‘true’, sebaliknya mengembalikan ‘false’.

e. Operasi Tambah Node
   - void insertDepan(const mahasiswa &data): Fungsi ini menambahkan node baru di depan linked list.
     - Membuat node baru dengan data mahasiswa yang diberikan.

- Jika list kosong, node baru menjadi head dan tail.
- Jika tidak kosong, node baru ditambahkan di depan dan pointer next dari node baru menunjuk ke head yang lama.

- void insertBelakang(const mahasiswa &data): Fungsi ini menambahkan node baru di belakang linked list.
  - Membuat node baru dengan data mahasiswa yang diberikan.
  - Jika list kosong, node baru menjadi head dan tail.
  - Jika tidak kosong, next dari node terakhir (tail) menunjuk ke node baru, dan kemudian tail diperbarui ke node baru.

f. Menghitung Jumlah Node

- int hitungList(): Fungsi ini menghitung dan mengembalikan jumlah node dalam linked list.
  - Mengiterasi dari head ke tail, dan menghitung setiap node yang ditemukan.

g. Menghapus Node

- void hapusDepan(): Fungsi ini menghapus node dari depan linked list.
  - Jika list tidak kosong, node yang ditunjuk oleh head dihapus, dan head diperbarui ke node berikutnya.
  - Jika setelah penghapusan, head menjadi nullptr, maka tail juga diatur ke nullptr, menandakan list kosong.
- void hapusBelakang(): Fungsi ini menghapus node dari belakang linked list.
  - Jika hanya ada satu node (yaitu head dan tail adalah node yang sama), maka setelah penghapusan, keduanya diatur ke nullptr.
  - Jika lebih dari satu node, fungsi ini mengiterasi hingga menemukan node sebelum tail untuk memperbarui tail.

h. Menampilkan Data

- void tampil(): Fungsi ini mencetak seluruh isi linked list.
  - Mengiterasi dari head ke tail dan mencetak nama serta NIM dari setiap node.
  - Jika list kosong, mencetak pesan bahwa list masih kosong.

i. Menghapus Seluruh List

- void clearList(): Fungsi ini menghapus seluruh node dalam linked list.
  - Mengiterasi dari head, menghapus setiap node satu per satu, dan akhirnya mengatur head dan tail ke nullptr.

Program ini mendemonstrasikan cara menggunakan single linked list untuk mengelola data mahasiswa. Dengan menggunakan struktur data ini, kita dapat menambah, menghapus, dan menampilkan mahasiswa dengan efisien. Output yang dihasilkan menunjukkan perubahan dalam linked list setelah setiap operasi, memungkinkan kita untuk melihat hasil dari setiap langkah.

2. guided_2

```
            }
            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
```

```
            }
            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!
```

Program ini digunakan untuk membuat dan mengelola sebuah linked list sederhana yang menyimpan nilai integer. Linked list adalah struktur data yang terdiri dari elemen-elemen (disebut node) yang saling terhubung satu sama lain. Program ini mencakup berbagai operasi dasar seperti menambahkan elemen, menampilkan elemen, menghitung jumlah elemen, dan menghapus semua elemen.

a. Definisi Struktur
   - Struct Node: Mewakili setiap elemen dalam linked list.
     - data: Menyimpan nilai integer yang menjadi data dari node.
     - next: Pointer yang menunjuk ke node berikutnya dalam linked list. Jika next adalah nullptr, itu berarti node ini adalah yang terakhir dalam list.

b. Fungsi Utama
   - Node* alokasi(int value):
     - Fungsi ini mengalokasikan memori untuk node baru.
     - Jika alokasi berhasil, fungsi ini mengisi data node dengan nilai yang diberikan dan mengatur next ke nullptr.
     - Fungsi mengembalikan pointer ke node baru yang telah dialokasikan.
   - void dealokasi(Node* node):
     - Fungsi ini mengembalikan memori yang digunakan oleh node ke sistem.
     - Ini dilakukan dengan menggunakan delete untuk membebaskan memori.

c. Pengecekan Kosong
   - bool isListEmpty(Node* head):
     - Fungsi ini memeriksa apakah linked list kosong dengan memeriksa apakah head adalah nullptr.
     - Jika head adalah nullptr, berarti tidak ada node dalam list.

d. Operasi Menambahkan Node

- void insertFirst(Node* &head, int value):
  - Fungsi ini menambahkan node baru di depan linked list.
  - Alokasikan node baru menggunakan fungsi alokasi(), dan jika berhasil, hubungkan node baru ke node yang ada saat ini (head) dan jadikan node baru sebagai head baru.
- void insertLast(Node* &head, int value):
  - Fungsi ini menambahkan node baru di akhir linked list.
  - Jika list kosong, node baru menjadi head.
  - Jika tidak kosong, fungsi ini mencari node terakhir dalam list dan menambahkan node baru di akhir.

e. Menampilkan Isi List
- void printList(Node* head):
  - Fungsi ini mencetak semua elemen dalam linked list.
  - Jika list kosong, mencetak pesan "List kosong!".
  - Jika tidak, fungsi ini akan mengiterasi setiap node dan mencetak data dari setiap node.

f. Menghitung Jumlah Elemen
- int countElements(Node* head):
  - Fungsi ini menghitung jumlah node dalam linked list.
  - Mengiterasi dari head sampai ke akhir list, menghitung setiap node yang ditemukan.

g. Menghapus Semua Elemen
- void clearList(Node* &head):
  - Fungsi ini menghapus semua elemen dalam linked list.
  - Mengiterasi setiap node, menyimpan pointer ke node saat ini, memindahkan pointer head ke node berikutnya, dan memanggil dealokasi() untuk membebaskan memori yang digunakan oleh node yang dihapus.

Program ini secara efektif mendemonstrasikan bagaimana menggunakan linked list untuk menyimpan dan mengelola data. Dengan menggunakan struktur data ini, kita dapat menambah, menghapus, dan menampilkan elemen dengan cara yang efisien dan dinamis. Jika ada bagian yang masih kurang jelas atau jika kamu ingin penjelasan lebih lanjut, silakan beri tahu!

IV. UNGUIDED

1. Task 1

Program ini adalah implementasi dari struktur data **linked list** (daftar berantai) dalam bahasa C++. Setiap node terdiri dari dua bagian: **data** (nilai) dan **pointer** (penunjuk) ke node berikutnya. Program ini memungkinkan kita untuk menambahkan elemen baru di depan atau di belakang daftar, serta mencetak isi dari linked list tersebut.

```
// Deklarasi Struct Node
struct Node {
    int data;
    Node* next;
};

// Deklarasi pointer untuk head dan tail
Node* head = nullptr;
Node* tail = nullptr;

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

    if (head == nullptr) {
        // Jika list kosong, node pertama menjadi head dan tail
        head = tail = baru;
    } else {
        // Jika list tidak kosong, node baru menjadi head
        baru->next = head;
        head = baru;
    }
}

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

    if (head == nullptr) {
        // Jika list kosong, node pertama menjadi head dan tail
        head = tail = baru;
    } else {
        // Jika list tidak kosong, node baru ditambahkan di belakang tail
        tail->next = baru;
        tail = baru;
    }
}
// Fungsi untuk mencetak seluruh isi linked list
void cetakList() {
    if (head == nullptr) {
        cout << "Linked list kosong.\n";
    } else {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << " -> ";
            }
            current = current->next;
        }
        cout << endl;
    }
}

// Main function
int main() {
    // Contoh penggunaan fungsi
    insertDepan(10);     // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5);      // Tambah node di depan (nilai: 5)

    // Cetak isi linked list
    cetakList();

    return 0;
```

- Komponen Utama dalam Program
  1. Node: Ini adalah unit dasar dari linked list. Setiap node berisi nilai data dan pointer yang mengarah ke node berikutnya. Jika tidak ada node berikutnya, pointer akan menjadi ‘nullptr’, menandakan akhir dari list.
  2. Head and Tail
     - Head adalah pointer yang menunjukkan awal dari linked list (node pertama).
     - Tail adalah pointer yang menunjukkan akhir dari linked list (node terakhir).
  3. Operasi-Operasi yang Tersedia
     - Menambah Node di Depan (Insert Depan): Menambahkan node baru di awal linked list. Node ini akan menjadi head baru dari list.
     - Menambah Node di Belakang (Insert Belakang): Menambahkan node baru di akhir linked list. Node ini akan menjadi tail baru dari list.
     - Mencetak Isi Linked List: Menampilkan semua elemen yang ada di linked list, mulai dari head hingga tail.

- Cara Kerja Program
  1. Menambah Node di Depan (Insert Depan):
     - Ketika kita ingin menambah node baru di depan list, kita buat node baru dengan nilai yang diberikan.
     - Jika linked list kosong (belum ada node), node baru ini akan menjadi head dan tail.
     - Jika linked list sudah berisi node, node baru akan ditempatkan di depan head, dan head akan digeser ke node baru ini.
  2. Menambah Node di Belakang (Insert Belakang):
     - Saat menambahkan node di belakang, kita buat node baru dengan nilai yang diberikan.
     - Jika linked list kosong, node baru ini akan menjadi head dan tail, sama seperti saat menambah node di depan.
     - Jika linked list sudah memiliki elemen, node baru ini akan ditempatkan di belakang tail, dan tail akan dipindahkan ke node baru ini.
  3. Mencetak Isi Linked List:
     - Untuk mencetak seluruh elemen, kita mulai dari head dan mengikuti pointer ke node berikutnya hingga kita mencapai akhir list.
     - Setiap nilai yang ditemukan akan ditampilkan, diikuti oleh panah ‘(->)’ jika masih ada node berikutnya.

- Contoh Jalannya Program
  Misalnya, kita memasukkan beberapa nilai ke dalam linked list dengan langkah-langkah berikut:
  1. **insertDepan (10)**: Menambahkan node dengan nilai 10 di depan. Sekarang,

linked list berisi 10.

2. **insertBelakang (20)**: Menambahkan node dengan nilai 20 di belakang. Linked list sekarang berisi 10 -> 20.
3. **insertDepan (5)**: Menambahkan node dengan nilai 5 di depan. Linked list sekarang menjadi 5 -> 10 -> 20.

Akhirnya, saat kita mencetak isi linked list, akan ditampilkan :

```
5 -> 10 -> 20
```

2. Task 2

Program ini adalah implementasi dari struktur data **linked list** dalam bahasa C++. Program ini memungkinkan kita untuk menambah elemen (node) di depan atau dibelakang linked list, menghapus node yang memiliki nilai tertentu, dan mencetak isi dari linked list:

```
    // Jika node yang dihapus adalah head
    if (head->data == nilai) {
        Node* hapus = head;
        head = head->next;
        delete hapus;
        return;
    }

    // Mencari node yang akan dihapus
    Node* current = head;
    Node* prev = nullptr;
    while (current != nullptr && current->data != nilai) {
        prev = current;
        current = current->next;
    }

    // Jika node ditemukan
    if (current != nullptr) {
        prev->next = current->next;
        // Jika node yang dihapus adalah tail, perbarui tail
        if (current == tail) {
            tail = prev;
        }
        delete current;
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList() {
    if (head == nullptr) {
        cout << "Linked list kosong.\n";
    } else {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << "->";
            }
            current = current->next;
        }
        cout << endl;
    }
}
```

```
    // Jika node yang dihapus adalah head
    if (head->data == nilai) {
        Node* hapus = head;
        head = head->next;
        delete hapus;
        return;
    }

    // Mencari node yang akan dihapus
    Node* current = head;
    Node* prev = nullptr;
    while (current != nullptr && current->data != nilai) {
        prev = current;
        current = current->next;
    }

    // Jika node ditemukan
    if (current != nullptr) {
        prev->next = current->next;
        // Jika node yang dihapus adalah tail, perbarui tail
        if (current == tail) {
            tail = prev;
        }
        delete current;
    }
}

// Fungsi untuk mencetak seluruh isi linked list
void cetakList() {
    if (head == nullptr) {
        cout << "Linked list kosong.\n";
    } else {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data;
            if (current->next != nullptr) {
                cout << "->";
            }
            current = current->next;
        }
        cout << endl;
    }
}

// Main function
int main() {
    // Contoh penggunaan fungsi
    insertDepan(10);     // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5);      // Tambah node di depan (nilai: 5)

    // Hapus node dengan nilai 10
    hapusNode(10);

    // Cetak isi linked list setelah penghapusan
    cetakList();

    return 0;
}
```

- Komponen Utama
  1. **Node**: Node adalah elemen dasar dari linked list, yang menyimpan dua hal:
     - **Data**: Nilai yang disimpan dalam node.
     - **Pointer ke Node Berikutnya**: Menunjuk ke node berikutnya dalam list, atau nullptr jika tidak ada node lagi.
  2. **Head dan Tail**:
     - **Head** menunjuk ke node pertama dalam linked list.
     - **Tail** menunjuk ke node terakhir dalam linked list.

- Fungsi Utama Program
  1. **Menambah Node di Depan (Insert Depan)**
     - Node baru ditambahkan di awal linked list.
     - Jika linked list masih kosong, node baru akan menjadi **head** dan **tail**.
     - Jika sudah ada node dalam list, node baru akan ditempatkan di depan, dan head akan diperbarui ke node baru ini.
  2. **Menambah Node di Belakang (Insert Belakang)**
     - Node baru ditambahkan di akhir linked list.
     - Jika linked list kosong, node baru menjadi **head** dan **tail**.
     - Jika sudah ada node dalam list, node baru ini ditempatkan setelah node terakhir (tail), dan tail diperbarui ke node baru ini.
  3. **Menghapus Node Tertentu (Hapus Node)**
     - Fungsi ini menghapus node dari linked list yang memiliki nilai tertentu.
     - Jika node yang ingin dihapus adalah **head**, maka head akan diperbarui ke node berikutnya.
     - Jika node yang ingin dihapus ada di tengah atau di akhir list, maka node tersebut akan dilepas dari rantai dengan memperbarui pointer dari node sebelumnya agar menunjuk langsung ke node setelah node yang dihapus.
     - Jika node yang dihapus adalah **tail**, maka tail akan diperbarui ke node sebelumnya.
  4. **Mencetak Isi Linked List**
     - Fungsi ini mencetak seluruh elemen dalam linked list, mulai dari **head** hingga **tail**, dan menampilkan isi list dalam bentuk urutan.

- Cara Kerjs Program dengan Contoh Penggunaan

  ```
  5->20
  ```

  1. **Tambah Node di Depan dengan nilai 10**
     - Karena linked list kosong, node dengan nilai 10 menjadi head dan tail.
     - Isi linked list: 10.
  2. **Tambah Node di Belakang dengan nilai 20**
     - Node dengan nilai 20 ditambahkan setelah node 10, menjadi node terakhir.
     - Isi linked list: 10 -> 20.
  3. **Tambah Node di Depan dengan nilai 5**
     - Node dengan nilai 5 ditempatkan di awal, sehingga menjadi head baru.
     - Isi linked list: 5 -> 10 -> 20.
  4. **Hapus Node dengan nilai 10**
     - Program mencari node dengan nilai 10 dan menghapusnya.
     - Pointer node sebelumnya (5) diperbarui untuk menunjuk langsung ke node setelah node 10 (node 20).
     - Isi linked list setelah penghapusan: 5 -> 20.

Pada akhir program, ketika kita mencetak isi linked list, hasilnya adalah 5 -> 20, yang

menunjukkan bahwa node dengan nilai 10 telah berhasil dihapus dari linked list. Program ini menunjukkan fleksibilitas dari linked list dalam menambah dan menghapus elemen, serta pengelolaan struktur datanya dengan efisie

3. Task 3

Program ini memungkinkan untuk melakukan beberapa operasi penting termasuk menambah elemen (node) di depan atau di belakang linked list, mencari node tertentu di dalam linked list, dan menghiung panjang atau jumlah elemen dalam linked list

```
// Deklarasi Struct Node
struct Node {
    int data;
    Node* next;
};

// Deklarasi pointer untuk head dan tail
Node* head = nullptr;
Node* tail = nullptr;

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

    if (head == nullptr) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

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

    if (head == nullptr) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

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

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

// Main function
int main() {
    // Contoh penggunaan fungsi
    insertDepan(10);     // Tambah node di depan (nilai: 10)
    insertBelakang(20); // Tambah node di belakang (nilai: 20)
    insertDepan(5);      // Tambah node di depan (nilai: 5)

    // Cari node dengan nilai 20
    int nilaiCari = 20;
    if (cariNode(nilaiCari)) {
        cout << "Node dengan nilai " << nilaiCari << " ditemukan.\n";
    } else {
        cout << "Node dengan nilai " << nilaiCari << " tidak ditemukan.\n";
    }

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

    return 0;
}
```

- Komponen Utama
  1. **Node**: Node adalah unit dasar dari linked list. Setiap node menyimpan dua hal:
     - **Data**: Nilai yang disimpan dalam node.
     - **Pointer ke Node Berikutnya**: Penunjuk yang menghubungkan node tersebut ke node berikutnya dalam list. Jika pointer ini nullptr, berarti itu adalah node terakhir dalam list.
  2. **Head dan Tail**:
     - **Head**: Menunjuk ke node pertama dalam linked list.
     - **Tail**: Menunjuk ke node terakhir dalam linked list.

- Fungsi-Fungsi Utama
  1. **Menambah Node di Depan (Insert Depan)**
     - Fungsi ini digunakan untuk menambah node baru di awal linked list.
     - Jika linked list kosong, node baru akan menjadi **head** dan **tail**.
     - Jika linked list sudah ada isinya, node baru akan ditempatkan di depan, dan head akan diperbarui ke node baru ini.
  2. **Menambah Node di Belakang (Insert Belakang)**
     - Fungsi ini digunakan untuk menambah node baru di akhir linked list.
     - Jika linked list kosong, node baru akan menjadi head dan tail.
     - Jika sudah ada node dalam list, node baru akan ditambahkan setelah node terakhir (tail), dan tail akan diperbarui ke node baru tersebut.
  3. **Mencari Node dengan Nilai Tertentu**
     - Fungsi ini melakukan pencarian nilai dalam linked list.
     - Program akan memulai dari head dan memeriksa setiap node satu per satu.
     - Jika ditemukan node yang memiliki nilai yang dicari, fungsi akan mengembalikan hasil bahwa node tersebut ditemukan.
     - Jika tidak ditemukan, fungsi akan menyatakan bahwa node tersebut tidak ada dalam linked list.
  4. **Menghitung Panjang Linked List**
     - Fungsi ini menghitung jumlah node dalam linked list, dimulai dari head hingga tail.
     - Program akan menghitung setiap node yang dilewati sampai mencapai akhir list.
     - Nilai akhir dari penghitung menunjukkan berapa banyak elemen atau node yang ada dalam linked list.

- Cara Kerja Program dengan Contoh Penggunaan

```
Node dengan nilai 20 ditemukan.
Panjang linked list: 3
```

  1. **Tambah Node di Depan dengan nilai 10**
     - Node dengan nilai 10 ditambahkan ke linked list.
     - Karena ini node pertama, node tersebut menjadi **head** dan **tail**.
     - Isi linked list saat ini: 10.

2. **Tambah Node di Belakang dengan nilai 20**
   - Node dengan nilai 20 ditambahkan di belakang node 10.
   - Node baru ini menjadi tail, dan isi linked list sekarang adalah: 10 -> 20.
3. **Tambah Node di Depan dengan nilai 5**
   - Node dengan nilai 5 ditambahkan di depan.
   - Node ini menjadi head baru, dan isi linked list menjadi: 5 -> 10 -> 20.
4. **Mencari Node dengan Nilai 20**
   - Program akan mencari nilai 20 di dalam linked list.
   - Dimulai dari head (5), kemudian ke node berikutnya (10), dan akhirnya menemukan node dengan nilai 20.
   - Program akan menampilkan bahwa node dengan nilai 20 ditemukan.
5. **Menghitung Panjang Linked List**
   - Program menghitung jumlah node dalam linked list dengan melintasi setiap node dari head hingga tail.
   - Hasil penghitungan menunjukkan ada 3 node di dalam linked list.

Program ini menunjukkan bagaimana linked list dapat digunakan untuk menyimpan, mencari, dan menghitung elemen dengan efisien menggunakan operasi yang tepat. Struktur data linked list sangat berguna dalam situasi di mana kita membutuhkan pengelolaan memori yang dinamis dan fleksibel.

V. KESIMPULAN

Praktikum ini berhasil mendemonstrasikan penggunaan single linked list sebagai salah satu struktur data dinamis yang fleksibel. Single linked list memungkinkan ukuran data berubah sesuai kebutuhan, berbeda dengan array yang memerlukan definisi ukuran tetap sejak awal. Operasi dasar seperti penambahan elemen di awal atau akhir, penghapusan elemen, serta penelusuran elemen dapat dilakukan secara efektif menggunakan pointer. Struktur ini juga memungkinkan pengelolaan memori yang lebih efisien karena elemen-elemen disimpan di lokasi memori yang tidak harus berurutan. Melalui praktikum ini, pengelolaan data mahasiswa dengan single linked list, termasuk penambahan, penghapusan, dan penelusuran data, telah dilakukan secara efisien. Implementasi program ini menunjukkan bahwa linked list sangat bermanfaat untuk aplikasi yang memerlukan struktur data dinamis dan manipulasi elemen secara fleksibel