LAPORAN PRAKTIKUM
Modul 1V
“Single Linked List (Bagian Pertama)”

Disusun Oleh:
Berlian Seva Astryana -2311104067
Kelas
SISE-07-02

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **Tujuan**

1.1 Memahami penggunaan linked list dengan pointer operator- operator dala program.
1.2 Memahami operasi-operasi dasar dalam linked list.
1.3 Membuat program dengan menggunakan linked list dengan prototype yang ada.

2 Landasan Teori

Sebuah *linked list* merupakan salah satu bentuk struktur data yang terdiri dari serangkaian elemen data yang saling berhubungan melalui pointer. Setiap elemen dalam *linked list*, yang disebut *node*, berisi dua bagian utama: bagian pertama menyimpan data, dan bagian kedua adalah pointer yang mengarah ke node berikutnya dalam urutan. Struktur data ini memiliki sifat dinamis, artinya dapat berkembang dan mengecil sesuai kebutuhan tanpa batasan jumlah elemen seperti pada array. Dengan demikian, *linked list* menawarkan fleksibilitas yang lebih dibandingkan dengan struktur data lain yang bersifat statis, seperti array, terutama dalam hal penambahan dan penghapusan elemen.

Linked list dapat diimplementasikan menggunakan pointer karena sifatnya yang dinamis. Pointer memungkinkan akses langsung ke memori dan pengelolaan alokasi memori secara efisien. Dalam bahasa C++, *pointer* mempermudah penanganan hubungan antar-elemen di dalam list, memungkinkan kita untuk menghubungkan satu node dengan node lainnya secara langsung. Ada beberapa jenis *linked list*, termasuk *single linked list*, *double linked list*, dan *circular linked list*, namun yang akan dibahas di sini adalah *single linked list*.

Pada *single linked list*, setiap node hanya berisi satu pointer yang menunjuk ke node berikutnya. Node terakhir dalam list ini menunjuk ke NULL, yang menandakan bahwa tidak ada node lain setelahnya. Operasi dasar pada *single linked list* mencakup penciptaan, penyisipan, penghapusan, penelusuran, pencarian, serta pembaruan elemen.

Salah satu operasi dasar adalah penciptaan list baru, yang dimulai dengan menginisialisasi pointer pertama (disebut *head* atau *first*) dengan nilai NULL untuk menandai bahwa list kosong. Kemudian, proses alokasi memori digunakan untuk membuat node baru ketika data ditambahkan ke dalam list. Fungsi alokasi memori ini sangat penting karena setiap kali elemen baru dimasukkan, memori harus dialokasikan secara dinamis untuk elemen tersebut. Pada bahasa C, alokasi memori dilakukan menggunakan fungsi `malloc()`, sedangkan pada C++ dapat digunakan `new` untuk tujuan yang sama.

Proses penyisipan elemen pada *single linked list* bisa dilakukan di berbagai posisi dalam list, seperti di awal, di akhir, atau di antara node tertentu. Metode penyisipan yang paling umum adalah *insert first* (penyisipan di awal), di mana elemen baru ditempatkan sebagai elemen pertama dari list, dan *insert last* (penyisipan di akhir), di mana elemen baru ditambahkan pada posisi terakhir dalam list. Pada penyisipan awal, pointer *next* dari elemen baru akan menunjuk ke elemen pertama yang sebelumnya ada di list, dan pointer *first* dari list akan di-update untuk menunjuk ke elemen baru tersebut.

Operasi penelusuran dan penampilan elemen dalam list dilakukan melalui proses iterasi dari node pertama hingga node terakhir. Setiap node dikunjungi secara sekuensial, dan data yang tersimpan di dalamnya ditampilkan. Jika diperlukan, list juga dapat diiterasi untuk mencari elemen dengan nilai tertentu. Dalam hal ini, proses pencarian dilakukan secara linear, dimulai

dari elemen pertama hingga elemen yang dicari ditemukan atau hingga akhir list jika elemen tersebut tidak ada.

Selain itu, terdapat operasi untuk menghapus elemen dari *linked list*. Penghapusan dapat dilakukan di awal list (*delete first*), di akhir list (*delete last*), atau setelah node tertentu (*delete after*). Proses penghapusan biasanya melibatkan perubahan pointer dari node sebelumnya agar tidak lagi menunjuk ke node yang akan dihapus, dan memori dari node yang dihapus harus dikembalikan ke sistem menggunakan fungsi *dealokasi*.

Pada prakteknya, setelah operasi-operasi dasar ini diimplementasikan, kita dapat mengembangkan program yang lebih kompleks yang memanfaatkan *single linked list*. Sebagai contoh, program dapat mengelola data mahasiswa yang terdiri dari nama dan NIM dalam sebuah list dinamis. Dengan menggunakan operasi-operasi pada list ini, kita dapat menambahkan, menghapus, atau mencari data mahasiswa dengan lebih fleksibel dibandingkan dengan struktur data statis.

Dalam penerapan pada C++, operasi-operasi ini biasanya dideklarasikan dalam header file (*.h) yang menyimpan definisi tipe data, struktur, dan prototipe fungsi. Implementasi operasinya dilakukan pada file *.cpp, yang akan mengatur bagaimana *linked list* tersebut bekerja dalam program. Dengan menggunakan pointer, program *single linked list* bisa menangani data dinamis secara lebih efisien dan fleksibel.

3 Guided

3.1 Guided 1

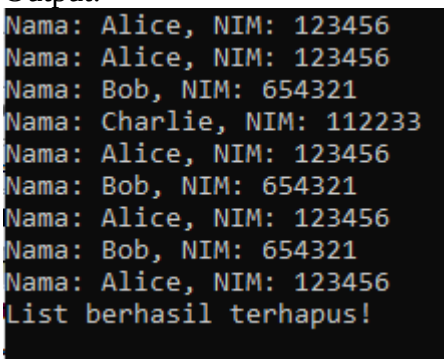
Program:

Telkom University Purwokerto

guided.cpp

```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;
}
```

Output:



3.2 Guided 2

Program:

Telkom University Purwokerto

guided2.cpp

```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;
}
```

4. Unguided

4.1 Membuat Single Linked List

Buatlah program C++ untuk membuat sebuah single linked list dengan operasi dasar sebagai berikut:

- Insert Node di Depan: Fungsi untuk menambah node baru di awal linked list.
- Insert Node di Belakang: Fungsi untuk menambah node baru di akhir linked list.
- Cetak Linked List: Fungsi untuk mencetak seluruh isi linked list.

Contoh input dan output:

Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cetak linked list

Output:

5 -> 10 -> 20

Program:

Telkom University Purwokerto

Unguided1.cpp

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

// Definisi Node untuk Linked List
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk menambah node di depan
void insertDepan(Node** head_ref, int new_data) {
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;

    // Hubungkan node baru ke head lama
    new_node->next = *head_ref;

    // Head sekarang menunjuk ke node baru
    *head_ref = new_node;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(Node** head_ref, int new_data)
{
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;
    new_node->next = nullptr;

    // Jika linked list kosong, jadikan node baru
sebagai head
    if (*head_ref == nullptr) {
        *head_ref = new_node;
        return;
    }

    // Jika tidak, cari node terakhir
    Node* last = *head_ref;
    while (last->next != nullptr) {
        last = last->next;
    }

    // Hubungkan node terakhir ke node baru
    last->next = new_node;
}

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

// Fungsi utama
int main() {
    // Head dari linked list diinisialisasi
sebagai null
    Node* head = nullptr;

    // Menambah node di depan dengan nilai 10
    insertDepan(&head, 10);

    // Menambah node di belakang dengan nilai 20
    insertBelakang(&head, 20);

    // Menambah node di depan dengan nilai 5
    insertDepan(&head, 5);

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

    return 0;
}
```

Output:

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

4.2 Menghapus Node pada Linked List

Buatlah program C++ yang dapat menghapus node tertentu dalam single linked list berdasarkan nilai yang diberikan oleh pengguna. Tugas ini mencakup operasi:

● Delete Node dengan Nilai Tertentu: Fungsi untuk menghapus node yang memiliki nilai tertentu.

● Cetak Linked List: Setelah penghapusan, cetak kembali isi linked list.

Contoh input/output:

Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Hapus node dengan nilai (nilai: 10)
5. Cetak linked list

Output:

5 -> 20

Program:

Telkom University Purwokerto

Unguided2.cpp

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

// Definisi Node untuk Linked List
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk menambah node di depan
void insertDepan(Node** head_ref, int new_data) {
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;

    // Hubungkan node baru ke head lama
    new_node->next = *head_ref;

    // Head sekarang menunjuk ke node baru
    *head_ref = new_node;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(Node** head_ref, int new_data)
{
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;
    new_node->next = nullptr;

    // Jika linked list kosong, jadikan node baru
sebagai head
    if (*head_ref == nullptr) {
        *head_ref = new_node;
        return;
    }

    // Jika tidak, cari node terakhir
    Node* last = *head_ref;
    while (last->next != nullptr) {
        last = last->next;
    }

    // Hubungkan node terakhir ke node baru
    last->next = new_node;
}

// Fungsi untuk menghapus node dengan nilai
tertentu
void hapusNode(Node** head_ref, int key) {
    // Simpan pointer ke head
    Node* temp = *head_ref;
    Node* prev = nullptr;

    // Jika head memiliki nilai yang ingin dihapus
    if (temp != nullptr && temp->data == key) {
        *head_ref = temp->next;  // Ubah head
        delete temp;  // Hapus node head
        return;
    }

    // Cari node dengan nilai tertentu dan simpan
node sebelumnya (prev)
    while (temp != nullptr && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // Jika nilai tidak ditemukan
    if (temp == nullptr) {
        cout << "Node dengan nilai " << key << "
tidak ditemukan." << endl;
        return;
    }

    // Hapus node
    prev->next = temp->next;
    delete temp;
}

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

// Fungsi utama
int main() {
    // Head dari linked list diinisialisasi
sebagai null
    Node* head = nullptr;

    // Menambah node di depan dengan nilai 10
    insertDepan(&head, 10);

    // Menambah node di belakang dengan nilai 20
    insertBelakang(&head, 20);

    // Menambah node di depan dengan nilai 5
    insertDepan(&head, 5);

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

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

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

    return 0;
}
```

Output:

```
Linked List sebelum penghapusan: 5 -> 10 -> 20
Linked List setelah penghapusan: 5 -> 20
```

4.3 Mencari dan Menghitung Panjang Linked List

Buatlah program C++ yang dapat melakukan operasi berikut:

● Cari Node dengan Nilai Tertentu: Fungsi untuk mencari apakah sebuah nilai ada di dalam linked list.

● Hitung Panjang Linked List: Fungsi untuk menghitung jumlah node yang ada di dalam linked list.

Contoh input/output:

Input:

1. Tambah node di depan (nilai: 10)
2. Tambah node di belakang (nilai: 20)
3. Tambah node di depan (nilai: 5)
4. Cari node dengan nilai 20
5. Cetak panjang linked list

Output:

Node dengan nilai 20 ditemukan.

Panjang linked list: 3

Program:

Telkom University Purwokerto

Unguided3.cpp

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

// Definisi Node untuk Linked List
struct Node {
    int data;
    Node* next;
};

// Fungsi untuk menambah node di depan
void insertDepan(Node** head_ref, int new_data) {
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;

    // Hubungkan node baru ke head lama
    new_node->next = *head_ref;

    // Head sekarang menunjuk ke node baru
    *head_ref = new_node;
}

// Fungsi untuk menambah node di belakang
void insertBelakang(Node** head_ref, int new_data)
{
    // Alokasi node baru
    Node* new_node = new Node();

    // Isi data node baru
    new_node->data = new_data;
    new_node->next = nullptr;

    // Jika linked list kosong, jadikan node baru
sebagai head
    if (*head_ref == nullptr) {
        *head_ref = new_node;
        return;
    }

    // Jika tidak, cari node terakhir
    Node* last = *head_ref;
    while (last->next != nullptr) {
        last = last->next;
    }

    // Hubungkan node terakhir ke node baru
    last->next = new_node;
}

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

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

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

// Fungsi utama
int main() {
    // Head dari linked list diinisialisasi
sebagai null
    Node* head = nullptr;

    // Menambah node di depan dengan nilai 10
    insertDepan(&head, 10);

    // Menambah node di belakang dengan nilai 20
    insertBelakang(&head, 20);

    // Menambah node di depan dengan nilai 5
    insertDepan(&head, 5);

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

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

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

    return 0;
}
```

Outut:

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

5. **Kesimpulan**

   Materi tentang Linked List telah memberikan pemahaman yang baik tentang bagaimana struktur data ini bekerja dan kapan sebaiknya digunakan. Saya mengerti bahwa Linked List sangat berguna ketika kita tidak memiliki batasan awal untuk jumlah data atau ketika sering melakukan operasi penambahan dan penghapusan di tengah data. Selain itu, saya juga memahami kelebihan dan kekurangan Linked List sehingga dapat membuat keputusan yang tepat dalam memilih struktur data yang paling sesuai untuk suatu aplikasi.

STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md