LAPORAN PRAKTIKUM

PERTEMUAN 4

Nama :

Razhendriya Vania Ramadhan Suganjarsarwat (2311104048)

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

- Memahami penggunaan **linked list** dengan pointer operator dalam program.
- Memahami operasi-operasi dasar dalam **linked list** seperti penciptaan, penyisipan, penghapusan, dan penelusuran elemen.
- Membuat program menggunakan **linked list** sesuai dengan prototype yang ada

II. LANDASAN TEORI

Linked List adalah struktur data yang fleksibel, terdiri dari serangkaian elemen yang saling terhubung menggunakan pointer. Elemen ini dinamakan Node, yang terdiri dari dua bagian: data dan pointer menuju elemen berikutnya.

Keunggulan linked list dibandingkan array adalah sifatnya yang dinamis, sehingga elemen dapat bertambah atau berkurang sesuai kebutuhan. Ada berbagai jenis linked list, seperti Single Linked List, Double Linked List, dan Circular Linked List. Modul ini fokus pada Single Linked List, di mana setiap node hanya memiliki satu pointer yang mengarah ke elemen berikutnya.

Operasi dasar pada linked list meliputi:

- **Penciptaan dan inisialisasi list.**
- **Penyisipan elemen di awal, akhir, atau setelah node tertentu.**
- **Penghapusan elemen dari awal, akhir, atau elemen tertentu.**
- **Penelusuran dan menampilkan elemen-elemen pada list**

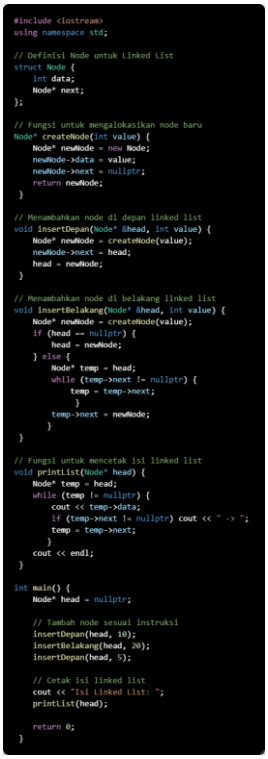
III. GUIDED

Program ini mendemonstrasikan cara membuat linked list, menambah node di awal dan akhir, menghitung jumlah node, menghapus node dari depan dan belakang, serta membersihkan seluruh list.

```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++;
        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;
}
```

Program ini menunjukkan cara kerja dasar linked list, termasuk menambahkan elemen di awal/akhir, menampilkan elemen, menghitung jumlah elemen, dan menghapus semua elemen dari list.

IV. UNGUIDED



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

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

// Fungsi untuk mengalokasikan node baru
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Menambahkan node di depan linked list
void insertDepan(Node* &head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

// Menambahkan node di belakang linked list
void insertBelakang(Node* &head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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

int main() {
    Node* head = nullptr;

    // Tambah node sesuai instruksi
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

    // Cetak isi linked list
    cout << "Isi Linked List: ";
    printList(head);

    return 0;
}
```

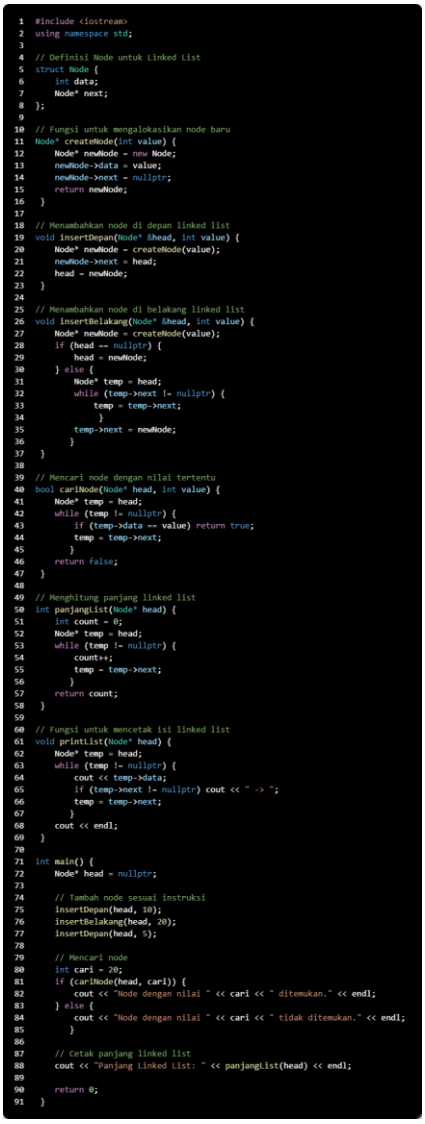
createNode: Mengalokasikan memori untuk node baru dan mengisinya dengan data yang diberikan.

insertDepan: Menyisipkan node di awal linked list dengan menghubungkan node baru ke head dan memperbarui head.

insertBelakang: Menambahkan node di akhir linked list dengan cara menelusuri node terakhir.

printList: Menampilkan elemen-elemen yang ada di dalam linked list.

deleteNode: Fungsi ini menghapus node dengan nilai tertentu. Jika node yang akan dihapus adalah node pertama, ia memperbarui head. Jika node ditemukan di bagian lain list, ia memutus hubungan node dan menghapus node tersebut dari memori.



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

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

// Fungsi untuk mengalokasikan node baru
Node* createNode(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;
    return newNode;
}

// Menambahkan node di depan linked list
void insertDepan(Node* &head, int value) {
    Node* newNode = createNode(value);
    newNode->next = head;
    head = newNode;
}

// Menambahkan node di belakang linked list
void insertBelakang(Node* &head, int value) {
    Node* newNode = createNode(value);
    if (head == nullptr) {
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}

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

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

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

int main() {
    Node* head = nullptr;

    // Tambah node sesuai instruksi
    insertDepan(head, 10);
    insertBelakang(head, 20);
    insertDepan(head, 5);

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

    // Cetak panjang linked list
    cout << "Panjang Linked List: " << panjangList(head) << endl;

    return 0;
}
```

cariNode: Fungsi untuk mencari apakah suatu node dengan nilai tertentu ada dalam linked list.

panjangList: Fungsi untuk menghitung jumlah node dalam linked list dengan cara menelusuri seluruh node.

V. KESIMPULAN

Single Linked List memungkinkan penyimpanan dan pengelolaan data dengan lebih efisien dan fleksibel. Dengan memahami operasi-operasi dasar seperti penyisipan, penghapusan, dan penelusuran, kita dapat memanfaatkan linked list untuk menyimpan data yang terus berkembang atau berubah ukurannya. Primitif-primitif yang diimplementasikan seperti insertFirst, insertLast, deleteFirst, deleteLast, dan printList adalah bagian penting dalam manipulasi linked list, dan penggunaannya sangat sesuai dengan kebutuhan program berbasis pointer