LAPORAN PRAKTIKUM

PERTEMUAN 5

Nama :

Razhendriya vania ramadhan suganjarsarwat (2311104048)

Dosen :

WAHYU ANDI SAPUTRA

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Tujuan:

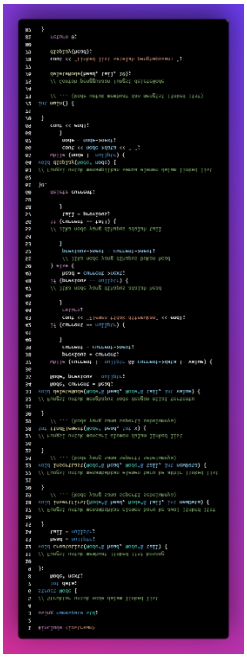
- Memahami penggunaan *linked list* dengan operator pointer dalam pemrograman.
- Memahami operasi-operasi dasar dalam *linked list*.
- Membuat program dengan menggunakan *linked list* berdasarkan prototipe yang ada

II. DASAR TEORI

Linked list **adalah struktur data yang terdiri dari elemen-elemen yang terhubung secara dinamis melalui pointer. Operasi dasar pada *linked list* mencakup *insertion*, *deletion*, dan pencarian (*searching*). Setiap elemen dalam *linked list* disebut *node*, yang berisi informasi (*info*) dan pointer ke elemen berikutnya.**

Proses pencarian (*searching*) dilakukan dengan mengunjungi setiap *node* hingga *node* yang dicari ditemukan, yang memudahkan operasi-operasi seperti *insert after*, *delete after*, dan *update*

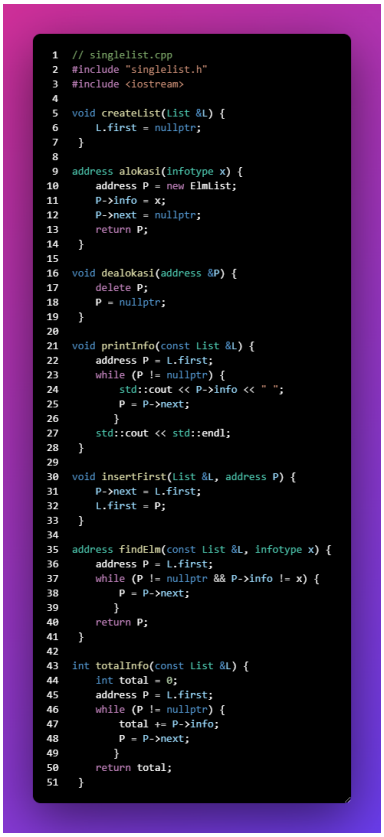
III. GUIDED



Kode C++ di atas mengimplementasikan struktur data linked list. Linked list adalah struktur data dinamis yang terdiri dari node-node, di mana setiap node berisi data dan pointer ke node berikutnya. Kode ini menyediakan fungsi-fungsi dasar untuk mengelola linked list, seperti membuat linked list kosong, menambahkan elemen di awal atau akhir, mencari elemen, menghapus elemen, dan menampilkan semua elemen. Fungsi deleteNode secara khusus mencari node dengan nilai tertentu dan menghapusnya dari linked list dengan memperbarui pointer pada node sebelumnya dan sesudahnya. Fungsi main memberikan contoh penggunaan fungsi deleteNode untuk menghapus elemen dengan nilai 10 dari linked list.

IV. UNGUIDED

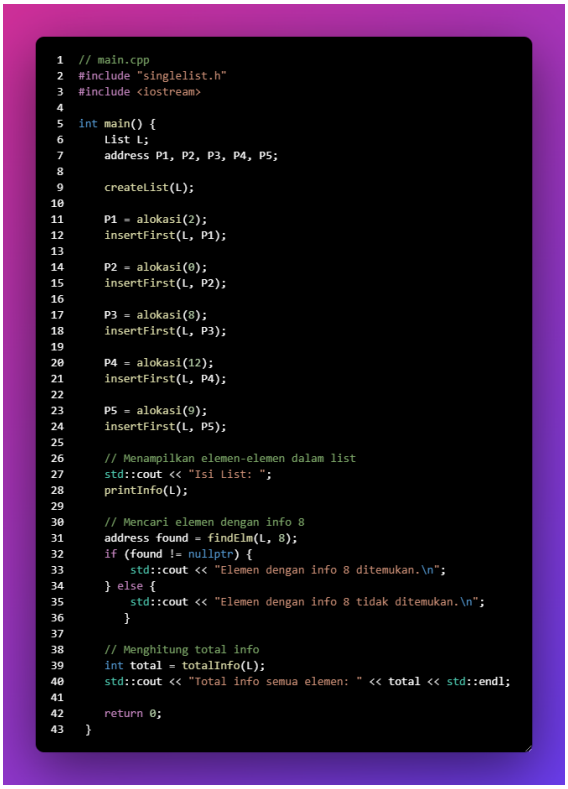
```h
// singlelist.h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);
address findElm(const List &L, infotype x);
int totalInfo(const List &L);

#endif
```



```cpp
// singlelist.cpp
#include "singlelist.h"
#include <iostream>

void createList(List &L) {
    L.first = nullptr;
}

address alokasi(infotype x) {
    address P = new ElmList;
    P->info = x;
    P->next = nullptr;
    return P;
}

void dealokasi(address &P) {
    delete P;
    P = nullptr;
}

void printInfo(const List &L) {
    address P = L.first;
    while (P != nullptr) {
        std::cout << P->info << " ";
        P = P->next;
       }
    std::cout << std::endl;
}

void insertFirst(List &L, address P) {
    P->next = L.first;
    L.first = P;
}

address findElm(const List &L, infotype x) {
    address P = L.first;
    while (P != nullptr && P->info != x) {
        P = P->next;
       }
    return P;
}

int totalInfo(const List &L) {
    int total = 0;
    address P = L.first;
    while (P != nullptr) {
        total += P->info;
        P = P->next;
       }
    return total;
}
```

1. **CreateList**: Menginisialisasi *linked list* kosong.
2. **alokasi**: Mengalokasikan node baru dengan nilai info.
3. **dealokasi**: Menghapus node dari memori.
4. **printInfo**: Menampilkan semua elemen dalam list.
5. **insertFirst**: Menambahkan elemen baru di awal list.

Implementasikan kode untuk setiap fungsi ini di singlelist.cpp agar *Single Linked List* dapat melakukan operasi dasar.



```cpp
// main.cpp
#include "singlelist.h"
#include <iostream>

int main() {
    List L;
    address P1, P2, P3, P4, P5;

    createList(L);

    P1 = alokasi(2);
    insertFirst(L, P1);

    P2 = alokasi(0);
    insertFirst(L, P2);

    P3 = alokasi(8);
    insertFirst(L, P3);

    P4 = alokasi(12);
    insertFirst(L, P4);

    P5 = alokasi(9);
    insertFirst(L, P5);

    // Menampilkan elemen-elemen dalam list
    std::cout << "Isi List: ";
    printInfo(L);

    // Mencari elemen dengan info 8
    address found = findElm(L, 8);
    if (found != nullptr) {
        std::cout << "Elemen dengan info 8 ditemukan.\n";
    } else {
        std::cout << "Elemen dengan info 8 tidak ditemukan.\n";
       }

    // Menghitung total info
    int total = totalInfo(L);
    std::cout << "Total info semua elemen: " << total << std::endl;

    return 0;
}
```

1. **Fungsi findElm**: Mencari node dengan nilai info tertentu (contohnya 8). Jika ditemukan, mengembalikan alamat node; jika tidak, mengembalikan nullptr.
2. **Fungsi totalInfo**: Menjumlahkan nilai info dari semua node dalam *linked list* dan mengembalikan hasil totalnya.

   Kedua fungsi ini mempermudah pencarian elemen dan perhitungan total nilai elemen dalam *linked list*.

V. KESIMPULAN

Dengan memahami konsep *linked list* dan berbagai operasinya, seperti pencarian, penambahan, dan penghapusan elemen, mahasiswa dapat membuat dan memanipulasi struktur data dinamis menggunakan bahasa pemrograman. Implementasi ini memungkinkan penanganan data secara efisien dalam aplikasi yang memerlukan manajemen data secara fleksibel