LAPORAN PRAKTIKUM

MODUL 5

SINGLE LINKED LIST (BAGIAN KEDUA)

Nama :

Candra Dinata (2311104061)

Kelas :

S1SE 07 02

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Membantu mahasiswa memahami dasar-dasar pemrograman, sepertisintaks, tipe data, operator, dan strukturkontrol, serta menguasaipenggunaan fungsi untuk modularitasprogram. Selain itu, mahasiswa jugadiperkenalkan dengan konsepObject-Oriented Programming (OOP) sepertikelas, objek, pewarisan, dan enkapsulasi. Melalui latihan ini, mahasiswadiharapkan mampu menerapkan konseppemrograman untuk menyelesaikanmasalah dan memahami carakerja kompilator C++ dalam mengubah kodesumber menjadiprogramyangdapatdijalankan.

II. LANDASAN TEORI

C++ adalah bahasa pemrograman yang dikembangkansebagaipengembangan dari bahasa C dengan menambahkan fitur-fiturpemrograman berorientasi objek (Object-OrientedProgramming, OOP). Dalam C++, konsep OOP seperti enkapsulasi, pewarisan, dan polimorfismemenjadi dasar untuk membuatprogram yang lebih modular, terstruktur, dandapat digunakankembali. Bahasa ini mendukung penggunaan fungsi, manipulasimemori secara langsung melalui pointer, sertamemilikikemampuan untuk menangani berbagai tipe data dasardan struktural. C++juga mendukung konsep pemrogramanprosedural dan modular, yangmemudahkan pengembanganprogram yang kompleks. Kompilasi programC++ dilakukanmelalui kompilator yang menerjemahkan kode sumbermenjadi fileeksekusi yang bisa dijalankanolehkomputer.

III. GUIDED

1.

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

struct Node {
    int data;
    Node* next;
};

void insertFirst(Node*& head, Node*& tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = head;
    head = new_node;

    if (tail == nullptr) {
        tail = new_node;
    }
}

void insertLast(Node*& head, Node*& tail, int new_data) {
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = head;
    head = new_node;

    if (tail == nullptr) {
        tail = new_node;
    }
}

int findElement(Node* head, int x) {
    Node* current = head;
    int index = 0;

    while (current != nullptr) {
        if (current->data == x) {
            return index;
        }
        current = current->next;
        index++;
    }
    return -1;
}

void display(Node* node) {
    while (node != nullptr) {
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}

void deleteElement(Node*& head, int x) {
    if (head == nullptr) {
        cout << "Linked list kosong" << endl;
        return;
    }

    // Jika elemen pertama adalah yang ingin dihapus
    if (head->data == x) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    // Cari elemen yang ingin dihapus
    Node* current = head;
    while (current->next != nullptr) {
        if (current->next->data == x) {
            Node* temp = current->next;
            current->next = current->next->next;
            delete temp;
            return;
        }
        current = current->next;
    }

    cout << "Elemen tidak ditemukan dalam linked list" << endl;
}

int main() {
    Node* head = nullptr;
    Node* tail = nullptr;

    insertFirst(head, tail, 11);
    insertFirst(head, tail, 14);
    insertFirst(head, tail, 18);
    insertFirst(head, tail, 3);
    insertFirst(head, tail, 5);
    insertFirst(head, tail, 7);

    cout << "Elemen dalam linked list: ";
    display(head);

    int x;
    cout << "Masukan elemen yang ingin dicari: ";
    cin >> x;

    int result = findElement(head, x);

    if (result == -1) {
        cout << "Elemen tidak ditemukan dalam linked list" << endl;
    } else {
        cout << "Elemen ditemukan pada indeks " << result << endl;
    }

    cout << "Masukan elemen yang ingin dihapus: ";
    cin >> x;
    deleteElement(head, x);

    cout << "Elemen dalam linked list setelah penghapusan: ";
    display(head);

    return 0;
}
```

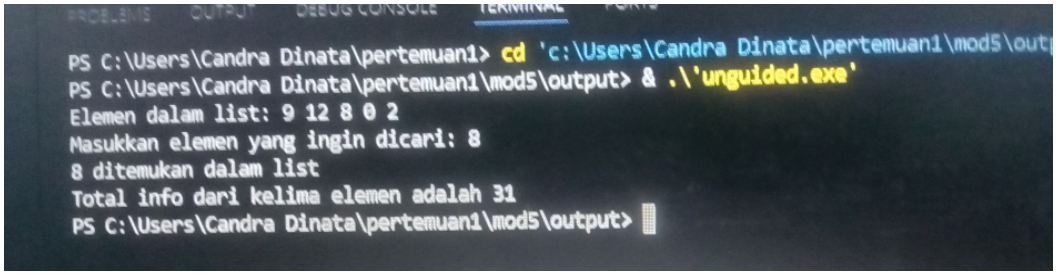
Kode ini adalah implementasi dari single linked list, yaitu struktur data linear di mana setiap elemen atau node menyimpan data dan pointer yang menunjuk ke node berikutnya. Struktur ini berguna untuk mengelola data yang tidak tetap jumlahnya karena linked list bisa bertambah atau berkurang ukurannya secara dinamis tanpa perlu mengalokasikan ulang memori secara keseluruhan. Di dalam kode ini, terdapat beberapa fungsi utama. Fungsi insertFirst menambahkan elemen baru di awal linked list, di mana elemen baru menjadi head yang menunjuk ke elemen sebelumnya sebagai node kedua. Ada juga findElement, yang melakukan pencarian posisi suatu elemen berdasarkan nilai yang diberikan; jika nilai ditemukan, posisi dikembalikan, dan jika tidak, hasil pencarian adalah -1. Fungsi display digunakan untuk menampilkan semua elemen di dalam linked list, mulai dari head hingga elemen terakhir.

Fungsi deleteElement menghapus node tertentu berdasarkan nilai yang diberikan. Jika elemen yang ingin dihapus adalah elemen pertama, fungsi akan memindahkan head ke elemen berikutnya. Namun, jika elemen berada di tengah atau akhir, fungsi akan mencari elemen tersebut, lalu memperbarui pointer next dari node sebelumnya untuk melewati node yang dihapus. Di dalam fungsi main, beberapa elemen ditambahkan ke dalam linked list melalui insertFirst, kemudian dilakukan proses pencarian dan penghapusan sesuai input pengguna. Kode ini juga menggunakan pointer head untuk menunjuk ke elemen pertama dalam linked list, dan pointer tail untuk menunjuk ke elemen terakhir, yang membantu memudahkan penambahan elemen di akhir. Dengan adanya fungsi-fungsi ini, single linked list bisa mengelola data secara efisien dalam struktur yang fleksibel, memungkinkan operasi penambahan, pencarian, penghapusan, dan penampilan data tanpa perlu menggeser elemen lain sebagaimana dalam array.

IV. UNGUIDED

1.

```cpp
#include <iostream>

using namespace std;

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

void createList(List &L) {
    L.First = NULL;
}

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

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

void printInfo(List L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

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

address findElm(List L, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info == x) {
            cout << x << " ditemukan dalam list" << endl;
            return P;
        }
        P = P->next;
    }
    cout << x << " tidak ditemukan dalam list" << endl;
    return NULL;
}

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

    // Insert elements
    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);

    // Print all elements
    cout << "Elemen dalam list: ";
    printInfo(L);

    // User input for element search
    infotype searchValue;
    cout << "Masukkan elemen yang ingin dicari: ";
    cin >> searchValue;

    // Find element
    findElm(L, searchValue);

    // Calculate total info of all elements
    address P = L.First;
    int totalInfo = 0;
    while (P != NULL) {
        totalInfo += P->info;
        P = P->next;
    }
    cout << "Total info dari kelima elemen adalah " << totalInfo << endl;

    return 0;
}
```



Kode ini adalah implementasi dasar dari struktur single linked list dalam C++, yang memungkinkan berbagai operasi dasar seperti membuat list, menambah elemen di awal, menampilkan elemen, mencari elemen, dan menghitung total nilai elemen di dalam list. Pada awalnya, fungsi createList menginisialisasi list baru dengan First bernilai NULL (list kosong). Fungsi alokasi membuat node baru yang berisi nilai tertentu (info) dan menunjuk next ke NULL, sedangkan dealokasi membebaskan memori dari node yang tidak diperlukan. Fungsi insertFirst menambahkan node baru di awal list dengan mengatur node tersebut sebagai First, dan menunjuk node berikutnya ke elemen pertama sebelumnya, sehingga membentuk rantai linked list. Fungsi printInfo menampilkan semua nilai dalam list, sementara findElm mencari elemen berdasarkan nilai, mengembalikan pointer node jika ditemukan dan mencetak pesan jika tidak ada. Di dalam main, beberapa elemen ditambahkan, dicetak, dan dicari sesuai masukan pengguna. Fungsi ini juga menghitung total nilai info dari semua elemen dalam list, memberikan gambaran tentang operasi dasar yang dapat dilakukan pada single linked list.

V. KESIMPULAN

Kesimpulan dari laporan praktikum terkait implementasi single linked list adalah bahwa struktur data ini memungkinkan pengelolaan data yang fleksibel dan dinamis. Dalam implementasinya, single linked list menggunakan pointer untuk membentuk hubungan antar node, di mana setiap node hanya menyimpan data dan pointer ke node berikutnya. Beberapa operasi dasar yang dapat dilakukan pada linked list ini mencakup inisialisasi list, menambah elemen di awal, mencari elemen berdasarkan nilai, mencetak seluruh elemen, serta menghitung total nilai elemen yang ada. Implementasi ini memanfaatkan fungsi createList untuk membuat list kosong, alokasi untuk mengalokasikan node baru, dan dealokasi untuk membebaskan memori. Fungsi-fungsi lain seperti insertFirst, printInfo, dan findElm membantu dalam manipulasi dan pencarian data di dalam list. Praktikum ini menunjukkan bagaimana single linked list dapat digunakan untuk menyimpan dan mengelola data secara dinamis, serta memberikan

pemahaman dasar tentang cara kerja struktur data linier yang efektif untuk operasi yang membutuhkan alokasi memori secara fleksibel.