LAPORAN PRAKTIKUM

PERTEMUAN 5

Disusun Oleh:

Naura Aisha Zahira (2311104078)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. **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.

2. **Landasan Teori**

Linked list adalah struktur data linear yang terdiri dari serangkaian elemen yang disebut "node," di mana setiap node berisi data dan pointer yang mengarah ke node berikutnya dalam urutan. Berbeda dengan array yang memiliki ukuran tetap, linked list memungkinkan ukuran dinamis dan fleksibel karena elemen-elemen dapat disisipkan atau dihapus dengan mudah tanpa harus menggeser elemen lainnya. Terdapat beberapa operasi dasar yang dapat dilakukan pada linked list, yaitu penambahan (insertion), pencarian (search), penghapusan (deletion), dan penampilan (display).

Dalam implementasi linked list, beberapa fungsi utama adalah:

- insertFirst - Menyisipkan elemen baru di awal linked list.
- insertLast - Menyisipkan elemen di akhir linked list.
- findElement - Mencari indeks suatu elemen berdasarkan nilai tertentu.
- display - Menampilkan semua elemen dari awal hingga akhir.
- deleteElement - Menghapus elemen tertentu dari linked list.

3. **Guided**

1) Struktur Node: Struktur Node merepresentasikan setiap elemen dalam linked list, yang memiliki dua anggota: data (menyimpan nilai elemen) dan next (pointer ke elemen berikutnya dalam list).

```
// Struktur untuk node dalama linked list
struct Node {
    int data;
    Node* next;
};
```

2) Fungsi insertFirst: Menambahkan elemen baru di awal linked list. Jika linked list kosong, elemen yang ditambahkan akan menjadi head dan tail sekaligus.

```
// Fungsi untuk menambahkan elemen baru ke awal linked list
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;
    }
}
```

3) Fungsi insertLast: Menambahkan elemen baru di akhir linked list. Jika linked list kosong, elemen baru akan menjadi head dan tail.

```
// FUngsi untuk menambahkan elemen baru ke akhir linked list
void insertLast(Node*& head, Node*& tail, int new_data){
    Node* new_node = new Node();
    new_node->data = new_data;
    new_node->next = nullptr;

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

4) Fungsi findElement: Mencari elemen dengan nilai tertentu di dalam linked list dan mengembalikan indeksnya jika ditemukan. Jika tidak ditemukan, fungsi mengembalikan -1.

```
// FUngsi untuk mencari elemen dalam linked list
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;
}
```

5) Fungsi display: Menampilkan semua elemen yang ada dalam linked list dari awal sampai akhir.

```
// Fungsi untuk menampilkan elemen dalam linked list
void display(Node* node){
    while (node != nullptr){
        cout << node->data << " ";
        node = node->next;
    }
    cout << endl;
}
```

6) Fungsi deleteElement: Menghapus elemen yang memiliki nilai tertentu dari linked list. Jika elemen yang dihapus adalah head, maka head diperbarui menjadi elemen berikutnya. Jika elemen berada di tengah atau akhir, fungsi menghapusnya dari linked list.

```
//Fungsi untuk menghapus elemen dari linked list
void deleteElement(Node*& head, int x){
    if (head == nullptr){
        cout << "Linked list kosong" << endl;
        return;
    }

    if (head->data == x){
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    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;
    }
}
```

7) Fungsi main:
    a. Membuat linked list kosong.
    b. Menambahkan beberapa elemen ke awal dan akhir linked list.
    c. Menampilkan semua elemen dalam linked list.
    d. Mencari elemen tertentu dalam linked list berdasarkan input pengguna.
    e. Menghapus elemen tertentu dari linked list berdasarkan input pengguna.
    f. Menampilkan linked list setelah penghapusan elemen.

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

    insertFirst(head, tail, 3);
    insertFirst(head, tail, 5);
    insertFirst(head, tail, 7);

    insertLast(head, tail, 11);
    insertLast(head, tail, 14);
    insertLast(head, tail, 18);

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

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

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

    return 0;
}
```

Output:

```
Elemen dalam linked list: 7 5 3 11 14 18
Masukkan elemen yang ingin dicari: 11
Elemen ditemukan pada indeks 3
Masukkan elemen yang ingin dihapus: 14
Elemen dalam linked list setelah penghapusan: 7 5 3 11 18
```

4. Unguided

1. singlelist.h

   Code:

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;

struct ElmtList {
    infotype info;
    ElmtList* next;
};

typedef ElmtList* address;

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 sumInfo(const List &L);

#endif
```

2. singlelist.cpp

```cpp
#include <iostream>
#include "singlelist.h"
using namespace std;

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

address alokasi(infotype x) {
    address P = new ElmtList;
    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) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << 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) {
        if (P->info == x) {
            return P;
        }
        P = P->next;
    }
    return nullptr;
}

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

3. main.cpp

```cpp
#include <iostream>
#include "singlelist.h"
using namespace std;

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

    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);

    printInfo(L);

    // Mencari elemen dengan info 8
    address found = findElm(L, 8);
    if (found != nullptr) {
         cout << "8 ditemukan dalam list" << endl;
    } else {
        cout << "8 tidak ditemukan dalam list" << endl;
    }

    // Menghitung total info dari seluruh elemen
    int total = sumInfo(L);
    cout << "Total info dari kelima elemen adalah " << total << endl;

    return 0;
}
```

Output:

```
9 12 8 0 2
8 ditemukan dalam list
Total info dari kelima elemen adalah 31
```

5. Kesimpulan

Dari percobaan ini, dapat disimpulkan bahwa linked list adalah struktur data yang fleksibel dan efisien untuk manipulasi data secara dinamis. Dengan menggunakan operasi dasar seperti penyisipan di awal atau akhir, pencarian, dan penghapusan, linked list memberikan cara yang efisien untuk mengelola data dalam program. Implementasi linked list ini membuktikan bahwa kita dapat dengan mudah menambah, mencari, dan menghapus elemen tanpa harus melakukan perubahan pada seluruh struktur data, sehingga memberikan keuntungan dibandingkan struktur data statis seperti array dalam kasus yang membutuhkan perubahan ukuran data secara dinamis.