LAPORAN PRAKTIKUM

MODUL

SINGLE LINKED LIST (BAGIAN KEDUA)

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. 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.

II. LANDASAN TEORI

1. *Linked List dengan Pointer*

linked list adalah salah satu struktur data yang terdiri dari serangkaian elemen yang saling terhubung, di mana setiap elemen memiliki data dan *pointer* yang menunjuk ke elemen berikutnya. Struktur ini fleksibel karena ukurannya dapat bertambah atau berkurang sesuai kebutuhan tanpa perlu mendefinisikan ukuran awal seperti pada array. Dalam implementasinya, *single linked list* hanya memerlukan satu arah *pointer*, sehingga setiap node hanya memiliki informasi untuk menghubungkan ke node berikutnya dan node terakhir menunjuk ke 'NULL' sebagai tanda akhir dari list.

Operasi dasar yang dapat dilakukan pada single linked list mencakup pembuatan list (*create*), penambahan elemen di awal (*insert first*) dan di akhir (*insert last*), penghapusan elemen (*delete*), serta penelusuran elemen (*traverse*). Penggunaan pointer memungkinkan *linked list* untuk menyimpan elemen-elemen secara dinamis di memori. Hal ini mempermudah penambahan atau penghapusan elemen di posisi tertentu dibandingkan array yang bersifat statis. Selain itu, *linked list* juga mendukung pencarian *sequntial* untuk menemukan elemen berdasarkan nilai tertentu.

Dalam program yang melibatkan *linked list*, fungsi 'cariNode' digunakan untuk mencari apakah suatu nilai ada dalam list, sedangkan 'hitungPanjang' bertujuan untuk menghitung jumlah node atau panjang dari list. Keunggulan dari *single linked list* adalah kemampuannya untuk mengelola memori secara efisien sesuai dengan kebutuhan data yang ada, menjadikannya solusi yang efektif untuk berbagai masalah yang memerlukan struktur data yang dapat berubah-ubah ukurannya.

III. GUIDED

Syntax :

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

// 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;
    }
}

// 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;
    }
}

// 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;
}

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

```
    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;
    }
}

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;
```

1. Struktur `Node`
   - `Node` adalah unit dasar dari linked list yang menyimpan dua informasi:
     - `data` yang menyimpan nilai.
     - `next` yang menyimpan alamat ke node berikutnya di dalam linked list.

2. Fungsi `insertFirst` (Menambahkan di awal linked list)

- Fungsi ini bertugas menambahkan elemen baru di awal linked list.
- Jika linked list kosong, maka elemen pertama juga akan menjadi elemen terakhir.
- Jika tidak kosong, elemen baru akan menjadi elemen pertama dan dihubungkan ke elemen lama yang sebelumnya berada di awal.

3. Fungsi `insertLast` (Menambahkan di akhir linked list)
- Fungsi ini menambahkan elemen baru di akhir linked list.
- Jika linked list kosong, elemen baru akan menjadi elemen pertama dan juga terakhir.
- Jika tidak, elemen baru akan dihubungkan di akhir list dan menjadi node terakhir.

4. Fungsi `findElement` (Mencari elemen dalam linked list)
- Fungsi ini mencari sebuah elemen dengan nilai tertentu dalam linked list.
- Fungsi ini melintasi setiap node dalam linked list, membandingkan nilai di setiap node dengan nilai yang dicari.
- Jika ditemukan, fungsi akan mengembalikan indeks elemen tersebut.
- Jika tidak ditemukan, fungsi akan mengembalikan nilai `-1`.

5. Fungsi `display` (Menampilkan elemen-elemen linked list)
- Fungsi ini melintasi seluruh node dalam linked list dan mencetak nilai dari setiap node satu per satu.

6. Fungsi `deleteElement` (Menghapus elemen dari linked list)
- Fungsi ini menghapus elemen dengan nilai tertentu dari linked list.
- Pertama, dicek apakah linked list kosong.
- Jika elemen yang akan dihapus adalah elemen pertama, maka pointer head akan dipindahkan ke elemen berikutnya dan elemen pertama dihapus.
- Jika elemen berada di tengah atau akhir, maka node sebelumnya akan dihubungkan dengan node setelah elemen yang dihapus, kemudian elemen tersebut dihapus dari memori.

7. Fungsi `main` (Program utama)
- Dalam fungsi ini, sebuah linked list dibangun dengan menambahkan elemen di awal (menggunakan `insertFirst`) dan di akhir (menggunakan `insertLast`).
- Linked list yang terbentuk kemudian ditampilkan menggunakan `display`.
- Pengguna diminta memasukkan elemen yang ingin dicari, dan hasil pencarian ditampilkan.
- Pengguna juga diminta memasukkan elemen yang ingin dihapus, kemudian linked list diperbarui dan ditampilkan kembali setelah penghapusan.

Alur program:
1. Linked list dibentuk.
2. Elemen-elemen ditambahkan di awal dan di akhir linked list.
3. Isi dari linked list ditampilkan.
4. Pengguna mencari sebuah elemen, dan hasil pencarian ditampilkan.

5. Pengguna menghapus sebuah elemen, dan linked list diperbarui serta ditampilkan kembali.

Program ini menunjukkan operasi dasar dalam linked list seperti menambah, mencari, dan menghapus elemen.

Output :

```
Elemen dalam linked list: 7 5 3 11 14 18
Masukkan elemen yang ingin dicari: 14
Elemen ditemukan pada indeks 4
Masukkan elemen yang ingin dihapus: 7
Elemen dalam linked list setelah penghapusan: 5 3 11 14 18
```

IV. UNGUIDED

1. Task

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct tElmtList *address;

struct tElmtList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Function prototypes
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address P);
void insertFirst(List &L, address P);
void printInfo(List L);
address findElm(List L, infotype x);
int sumElements(List L);

#endif
```

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

using namespace std;

// Create an empty list
void createList(List &L) {
    L.First = NULL;
}

// Allocate a new element
address alokasi(infotype x) {
    address P = new tElmtList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

// Deallocate a node
void dealokasi(address P) {
    delete P;
}

// Insert an element at the beginning of the list
void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}

// Print all elements in the list
void printInfo(List L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

// Find an element in the list
address findElm(List L, infotype x) {
    address P = L.First;
    while (P != NULL) {
        if (P->info == x) {
            return P; // Element found
        }
// Sum the info values of all elements in the list
int sumElements(List L) {
    address P = L.First;
    int sum = 0;
    while (P != NULL) {
        sum += P->info;
        P = P->next;
    }
    return sum;
}
```

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

using namespace std;

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

    // Inisialisasi list
    createList(L);

    // Masukkan elemen-elemen ke dalam list
    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);

    // Menanyakan pilihan operasi kepada pengguna
    cout << "Pilih opsi:\n";
    cout << "1. Tampilkan elemen-elemen dalam list\n";
    cout << "2. Cari elemen dengan nilai info 8\n";
    cout << "3. Hitung total dari semua elemen\n";
    cout << "Masukkan pilihan Anda (1-3): ";
    cin >> pilihan;

    switch (pilihan) {
        case 1:
            // Menampilkan elemen-elemen dalam list
            cout << "Elemen dalam list: ";
            printInfo(L);
            break;
```

```
            case 2:
                // Mencari elemen dengan nilai info 8
                {
                    address ditemukan = findElm(L, 8);
                    if (ditemukan != NULL) {
                        cout << ditemukan->info << " ditemukan dalam list" << endl;
                    } else {
                        cout << "Elemen tidak ditemukan" << endl;
                    }
                }
                break;

            case 3:
                // Menghitung dan menampilkan total dari semua elemen
                int total;
                total = sumElements(L);
                cout << "Total info dari kelima elemen adalah " << total << endl;
                break;

            default:
                cout << "Pilihan tidak valid!" << endl;
        }

        return 0;
    }
```

penjelasan detail mengenai komponen-komponen program:

1. Struktur dan Tipe Data

- **infotype**: Didefinisikan sebagai tipe data int. Ini digunakan untuk menyimpan nilai informasi dari setiap elemen di dalam linked list.
- **address**: Ini adalah pointer ke sebuah tElmtList, yang mengacu pada node atau elemen di dalam linked list.
- **tElmtList**: Struktur yang mewakili satu node dalam linked list. Struktur ini memiliki dua anggota:
    - info: Menyimpan nilai elemen (dari tipe infotype, yaitu int).
    - next: Menyimpan pointer yang menunjuk ke elemen berikutnya dalam linked list.
- **List**: Struktur yang digunakan untuk menyimpan pointer First, yaitu pointer ke elemen pertama dalam linked list. Ini digunakan untuk melacak elemen awal dari linked list.

2. Fungsi-fungsi Utama

a. createList(List &L)

- Fungsi ini bertujuan untuk menginisialisasi linked list agar kosong pada awalnya. Dalam hal ini, First pada list diatur menjadi NULL, yang menunjukkan bahwa tidak ada elemen yang disimpan di dalam linked list.

b. alokasi(infotype x)

- Fungsi ini membuat elemen baru di dalam linked list. Sebuah elemen baru dialokasikan dengan menggunakan operator new untuk menciptakan node. Nilai yang diberikan (x) disimpan dalam info elemen tersebut, dan pointer next diatur

menjadi NULL (karena elemen tersebut belum dihubungkan ke elemen lain).

c. dealokasi(address P)

- Fungsi ini digunakan untuk melepaskan atau menghapus elemen tertentu dari memori. Operator delete digunakan untuk menghapus node yang ditunjuk oleh pointer P.

d. insertFirst(List &L, address P)

- Fungsi ini menambahkan elemen baru ke awal linked list. Elemen baru yang dialokasikan akan menjadi elemen pertama. Sebelum ini, elemen baru dihubungkan dengan elemen sebelumnya yang berada di awal list. Setelah itu, pointer First diatur untuk menunjuk ke elemen baru.

e. printInfo(List L)

- Fungsi ini digunakan untuk menampilkan semua elemen dalam linked list. Fungsi ini mengakses elemen pertama, kemudian mencetak nilai info dari setiap elemen satu per satu hingga mencapai elemen terakhir (NULL).

f. findElm(List L, infotype x)

- Fungsi ini mencari elemen di dalam linked list berdasarkan nilai info yang diberikan. Fungsi akan memeriksa setiap elemen dalam linked list dan membandingkan nilai info dengan nilai x. Jika ditemukan, fungsi akan mengembalikan pointer ke elemen tersebut. Jika tidak, fungsi mengembalikan NULL.

g. sumElements(List L)

- Fungsi ini menghitung total nilai info dari semua elemen yang ada dalam linked list. Fungsi akan melintasi setiap elemen, menambahkan nilai info ke dalam sebuah variabel total hingga mencapai akhir list.

3. Fungsi main (Program Utama)

- Program utama dimulai dengan menginisialisasi sebuah linked list yang kosong menggunakan createList.
- Kemudian, lima elemen dimasukkan ke dalam linked list menggunakan insertFirst. Elemen-elemen ini memiliki nilai: 2, 0, 8, 12, dan 9, dan mereka akan ditambahkan di awal list, sehingga urutannya menjadi [9, 12, 8, 0, 2].
- Program selanjutnya menampilkan pilihan kepada pengguna:
    - **Pilihan 1**: Menampilkan semua elemen di dalam linked list.
    - **Pilihan 2**: Mencari elemen dengan nilai info 8.
    - **Pilihan 3**: Menghitung dan menampilkan total dari semua elemen dalam linked list.
- Pengguna akan memasukkan pilihan mereka, dan program akan menjalankan operasi yang sesuai berdasarkan pilihan yang dimasukkan.
    - Jika pengguna memilih untuk menampilkan elemen (Pilihan 1), program akan menggunakan fungsi printInfo untuk mencetak semua nilai elemen yang ada di dalam linked list.
    - Jika pengguna memilih untuk mencari elemen dengan nilai 8 (Pilihan 2), program akan menggunakan fungsi findElm untuk menemukan elemen tersebut dan menampilkan pesan jika elemen ditemukan.
    - Jika pengguna memilih untuk menghitung total elemen (Pilihan 3), program

akan menggunakan fungsi sumElements untuk menghitung total dari semua nilai elemen dalam linked list dan menampilkannya.

4. Logika Operasi

- Program ini mengimplementasikan dasar-dasar operasi pada linked list seperti:
  - Menambah elemen di awal linked list.
  - Mencari elemen tertentu.
  - Mencetak isi dari linked list.
  - Menghitung total dari semua elemen yang ada.
- Setiap operasi dilakukan dengan cara melintasi linked list menggunakan pointer untuk mengakses node satu per satu, dari node pertama hingga akhir.

Program ini memberikan gambaran yang baik tentang bagaimana linked list bekerja, termasuk bagaimana elemen-elemen disimpan secara dinamis, bagaimana elemen baru ditambahkan, dan bagaimana kita bisa mencari atau mengoperasikan elemen-elemen dalam list tersebut.

Output :

```
Pilih opsi:
1. Tampilkan elemen-elemen dalam list
2. Cari elemen dengan nilai info 8
3. Hitung total dari semua elemen
Masukkan pilihan Anda (1-3): 1
Elemen dalam list: 9 12 8 0 2
```

```
Pilih opsi:
1. Tampilkan elemen-elemen dalam list
2. Cari elemen dengan nilai info 8
3. Hitung total dari semua elemen
Masukkan pilihan Anda (1-3): 2
8 ditemukan dalam list
```

```
Pilih opsi:
1. Tampilkan elemen-elemen dalam list
2. Cari elemen dengan nilai info 8
3. Hitung total dari semua elemen
Masukkan pilihan Anda (1-3): 3
Total info dari kelima elemen adalah 31
```

V. KESIMPULAN

Dalam praktikum ini, kami belajar bagaimana menggunakan **single linked list**, sebuah struktur data dinamis yang memungkinkan penyimpanan data secara terhubung. Linked list dimulai dengan kondisi kosong, lalu elemen-elemen baru bisa ditambahkan satu per satu menggunakan alokasi memori secara dinamis. Salah satu keunggulannya adalah kita dapat menambah atau menghapus elemen dengan mudah tanpa harus menggeser elemen lain, seperti yang biasa terjadi pada array. Namun, pencarian elemen

membutuhkan kita menelusuri dari elemen pertama sampai yang dicari, sehingga kurang efisien untuk ukuran data yang besar. Meski begitu, linked list tetap sangat bermanfaat untuk skenario yang membutuhkan penambahan dan penghapusan elemen yang cepat. Praktikum ini membantu kami memahami bagaimana linked list bekerja, serta pentingnya pengelolaan memori secara efisien dalam program yang dinamis.elemen disimpan di lokasi memori yang tidak harus berurutan. Melalui praktikum ini, pengelolaan data mahasiswa dengan single linked list, termasuk penambahan, penghapusan, dan penelusuran data, telah dilakukan secara efisien. Implementasi program ini menunjukkan bahwa linked list sangat bermanfaat untuk aplikasi yang memerlukan struktur data dinamis dan manipulasi elemen secara fleksibel.