LAPORAN PRAKTIKUM
MODUL 5
SINGLE LINKED LIST (BAGIAN KEDUA)

Disusun Oleh:

Satria Putra Dharma Prayudha - 21104036

SE07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan

1. Memahami penggunaan linked list dengan pointer operator- operator dalam program.
2. Memahami operasi-operasi dasar dalam linked list.
3. Membuat program dengan menggunakan linked list dengan prototype yang ada.

B. Landasan Teori

Landasan teori ini berdasarkan pada modul pembelajaran praktikum struktur data kali ini

2.1 Searching

Searching adalah salah satu operasi dasar pada struktur data Single Linked List. Operasi ini bertujuan untuk mencari *node* tertentu dengan cara mengunjungi setiap *node* satu per satu hingga *node* yang dicari ditemukan. Searching memiliki peran penting dalam mempermudah pelaksanaan operasi lain seperti *insert after*, *delete after*, dan *update*.

Setiap operasi dasar dalam Single Linked List, termasuk *searching*, merupakan bagian dari *Abstract Data Type* (ADT). Dalam implementasi menggunakan bahasa pemrograman seperti C, ADT ini disimpan dalam file header (*.h) dan file implementasi (*.c).

Operasi pencarian dilakukan dengan fungsi-fungsi berikut:

- findElm: Mencari elemen dengan nilai tertentu. Fungsi ini mengembalikan alamat elemen jika ditemukan, atau Nil jika elemen tidak ditemukan.
- fFindElm: Mencari elemen dengan alamat tertentu, mengembalikan *true* jika alamat ditemukan, dan *false* jika tidak.
- findBefore: Mengembalikan alamat elemen sebelum elemen yang dicari.

Dengan adanya operasi *searching* ini, program dapat mengelola data dalam linked list dengan lebih efektif dan efisien.

Guided

a. Searching

Code :

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

// Sturktur untuk node dalam linked list

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

// Fungsi untuk menambahkan elemen baru di awal linked
list
void insertFirst(Node*& head, Node*& tail, int
new_data) {
    Node* newNode = new Node();
    newNode->data = new_data;
    newNode->next = head;
    head = newNode;

    if (tail == NULL) {
        tail = head;
    }
}

// Fungsi untuk menambahkan elemen baru di akhir linked
list
void insertLast(Node*& head, Node*& tail, int new_data)
{
    Node* newNode = new Node();
    newNode->data = new_data;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        tail = newNode;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
}

// Fungsi untuk mencari elemen dalam linked list

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

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

    return -1;
}

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

// Fungsi untuk menghapus elemen dalam linked list
void deleteElement(Node*& head, int x) {
    if(head == NULL) {
        cout << "List is empty" << endl;
        return;
    }

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

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

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

    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 << "Elements in linked list: ";
    display(head);

    int x;
    cout << "Enter element to be searched: ";
    cin >> x;

    int result = findELement(head, x);

    if(result == -1) {
        cout << "Element not found" << endl;
    } else {
        cout << "Element found at index: " << result <<
endl;

    cout << "Enter element to be deleted: ";
    cin >> x;
    deleteElement(head, x);

    cout << "Elements in linked list after deletion: ";
    display(head);

    return 0;
    }
}
```

Output :

```
nerFile.cpp -o tempCodeRunnerFile } ; if ($?) { .\tempCodeRunnerFile }
Elements in linked list: 7 5 3 11 14 18
Enter element to be searched: 5
Element found at index: 1
Enter element to be deleted: 11
Elements in linked list after deletion: 7 5 3 14 18
```

Penjelasan : Program di atas mengimplementasikan *Single Linked List* dalam C++ dengan beberapa fungsi dasar yaitu menambahkan, mencari, menampilkan, dan menghapus elemen dalam *linked list*. Untuk penjelasan lebih singkatnya :

1. Struktur Node: Setiap *node* dalam *linked list* memiliki dua atribut, yaitu data untuk menyimpan nilai elemen, dan next sebagai pointer yang menunjuk ke node berikutnya. Ini mendefinisikan hubungan antar node dalam linked list.
2. Fungsi insertFirst: Fungsi ini digunakan untuk menambahkan elemen baru di awal linked list. Node baru akan menjadi *head*, dan jika linked list sebelumnya kosong, maka tail juga diatur untuk menunjuk ke node pertama.
3. Fungsi insertLast: Fungsi ini menambahkan elemen baru di akhir linked list. Jika linked list masih kosong, node baru menjadi head sekaligus tail. Jika tidak, tail diperbarui untuk menunjuk ke node baru.
4. Fungsi findElement: Fungsi ini mencari elemen tertentu dalam linked list berdasarkan nilai yang dicari. Jika elemen ditemukan, fungsi mengembalikan indeks posisi elemen tersebut. Jika tidak, fungsi mengembalikan nilai -1 yang menandakan elemen tidak ditemukan.
5. Fungsi display: Fungsi ini digunakan untuk menampilkan seluruh elemen dalam linked list secara berurutan dari awal (head) hingga akhir.
6. Fungsi deleteElement: Fungsi ini menghapus elemen dari linked list. Jika elemen yang dihapus berada di posisi pertama, head diperbarui. Jika berada di posisi lain, pointer node sebelumnya diatur untuk melewati node yang akan dihapus.

Pada fungsi main, program dimulai dengan menambahkan beberapa elemen di awal dan akhir linked list, kemudian menampilkan isinya. Setelah itu, pengguna dapat mencari elemen berdasarkan input dan menghapus elemen tertentu yang juga dimasukkan oleh pengguna.

C. Unguided

Pada kode bagian unguided ini terdapat main.cpp dan singlelist.h yang digunakan untuk me membuat dan menjalankan program :

Main.cpp :

```cpp
#include "singlelist.h"

int main() {
    List L;
    createList(L);  // Membuat list kosong

    // Alokasi dan masukkan beberapa elemen ke dalam
list
    address P1 = alokasi(2);
    address P2 = alokasi(0);
    address P3 = alokasi(8);
    address P4 = alokasi(12);
    address P5 = alokasi(9);

    // Menambahkan elemen-elemen tersebut ke dalam list
    insertFirst(L, P1);  // Menambahkan elemen 2
    insertFirst(L, P2);  // Menambahkan elemen 0
    insertFirst(L, P3);  // Menambahkan elemen 8
    insertFirst(L, P4);  // Menambahkan elemen 12
    insertFirst(L, P5);  // Menambahkan elemen 9

    // Cetak semua elemen di dalam list
    cout << "Isi dari Linked List: ";
    printInfo(L);  // Output: 9 12 8 0 2

    return 0;
}
```

Penjelasan :

- Setiap soal memiliki file main.cpp terpisah untuk menguji berbagai operasi. Program utama di setiap file akan mengalokasikan elemen-elemen, memasukkan elemen ke dalam list, dan menampilkan hasil sesuai tujuan latihan.

Singlelist.h :

```h
#ifndef SINGLELIST_H
#define SINGLELIST_H

#include <iostream>
using namespace std;

typedef int infotype;  // Tipe data info
typedef struct elmlist *address;  // Alamat dari elemen
dalam linked list

// Struktur elemen dari Single Linked List
struct elmlist {
    infotype info;
    address next;
};

// Struktur list
struct List {
    address first;  // Elemen pertama dari list
};

// Deklarasi fungsi-fungsi dasar untuk operasi pada
Single Linked List
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);

#endif
```

Penjelasan :

- Berisi implementasi dari semua fungsi yang dideklarasikan di singlelist.h.
- Fungsi ini meliputi pembuatan list kosong, alokasi dan dealokasi memori, pencarian elemen, dan penyisipan elemen di awal list.fungsi-fungsi yang akan kita implementasikan di singlelist.cpp.

Latihan dibawah ini merupakan file singlelist.cpp yang akan dibuat dan disesuaikan dengan kebutuhan serta penyesuaian pada file lainnya :

a. Membuat Single Linked List

Code:

```cpp
#include "singlelist.h"

// Membuat list kosong
void createList(List &L) {
    L.first = nullptr;  // List awalnya kosong
}

// Alokasi elemen baru
address alokasi(infotype x) {
    address P = new elmlist;  // Membuat elemen baru
    if (P != nullptr) {
        P->info = x;  // Mengisi info dengan nilai x
        P->next = nullptr;  // Elemen baru belum terhubung ke elemen lain
    }
    return P;
}

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

// Mencetak info dari seluruh elemen list
void printInfo(const List &L) {
    address P = L.first;
    while (P != nullptr) {
        cout << P->info << " ";  // Cetak info dari elemen
        P = P->next;  // Lanjut ke elemen berikutnya
    }
    cout << endl;
}

// Menyisipkan elemen di awal list
void insertFirst(List &L, address P) {
    P->next = L.first;  // Elemen baru menunjuk ke elemen pertama
    L.first = P;  // Elemen pertama sekarang adalah elemen baru
}
```

Output:

```
PS D:\Kuliah\Struktur Data\Github\0
Isi dari Linked List: 9 12 8 0 2
PS D:\Kuliah\Struktur Data\Github\0
```

Penjelasan:

- Fungsi createList membuat list kosong dengan menginisialisasi pointer first menjadi nullptr.
- Setiap elemen baru dialokasikan dengan alokasi() yang menciptakan node baru dan mengisi informasi elemen (info).
- Fungsi insertFirst digunakan untuk menyisipkan elemen-elemen tersebut di awal list, sehingga elemen terbaru selalu berada di posisi terdepan.
- Fungsi printInfo mencetak semua elemen dalam urutan penyisipan. Pada latihan ini, list akan berisi: 9, 12, 8, 0, 2.

-

b. Mencari Elemen dengan Info 8

Code:

Pada Main.cpp ditambahkan :

```
// Mencari elemen dengan info 8
address result = findElm(L, 8);
if (result != nullptr) {
    cout << "Elemen dengan info 8 ditemukan." << endl;
} else {
    cout << "Elemen dengan info 8 tidak ditemukan." << endl;
}
```

Pada SingleList.h ditambahkan :

```
address findElm(const List &L, infotype x);
```

Pada SingleList.cpp ditambahkan :

```
address findElm(const List &L, infotype x) {
    address P = L.first;
    while (P != nullptr) {
        if (P->info == x) {
            return P;  // Jika ditemukan, kembalikan alamat elemen
        }
        P = P->next;  // Lanjut ke elemen berikutnya
    }
    return nullptr;  // Jika tidak ditemukan, kembalikan nullptr
}
```

Output:

```
PS D:\Kuliah\Struktur Data\Github\05
Elemen dengan info 8 ditemukan.
PS D:\Kuliah\Struktur Data\Github\05
```

Penjelasan:

- Setelah membuat list dan menambahkan elemen-elemen seperti pada Latihan 2, fungsi baru findElm dipanggil.
- findElm mencari node dengan nilai tertentu (contohnya 8) dengan cara traversing node satu per satu mulai dari first.
- Jika elemen ditemukan, fungsi mengembalikan alamat node tersebut; jika tidak, fungsi mengembalikan nullptr.
- Dalam main program, hasil pencarian akan ditampilkan di layar.

c. Hitung Jumlah Total Info Seluruh Elemen

Code:

Pada Main.cpp ditambahkan :

```
// Menghitung total nilai dari seluruh elemen
  int total = totalInfo(L);
  cout << "Total nilai info elemen: " << total << endl;  // Output: 31
```

Pada SingleList.h ditambahkan :

```
int totalInfo(const List &L);
```

Pada SingleList.cpp ditambahkan :

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

Output:

```
PS D:\Kuliah\Struktur Data\Git
Total nilai info elemen: 31
PS D:\Kuliah\Struktur Data\Git
```

Penjelasan:

1. Sama seperti latihan sebelumnya, list dibuat dan diisi dengan elemen-elemen menggunakan insertFirst.
2. Fungsi totalInfo digunakan untuk menghitung jumlah seluruh nilai

dalam list dengan menelusuri setiap elemen menggunakan pointer.

3. Setiap nilai dalam node akan dijumlahkan dan hasil akhirnya dicetak di layar.
4. Pada contoh ini, nilai yang dijumlahkan adalah: 9 + 12 + 8 + 0 + 2 = 31.

D. Kesimpulan

Pada modul praktikum ini, telah dipelajari konsep dasar dan implementasi Single Linked List menggunakan bahasa pemrograman C++. Dalam modul ini dapat suatu hal yaitu memahami penggunaan pointer dan operasi dasar seperti penyisipan, pencarian, dan penghapusan elemen, yang menunjukkan fleksibilitas struktur data ini dalam mengelola data secara dinamis. Melalui operasi searching, dan dari modul ini belajar cara mencari elemen tertentu dalam linked list menggunakan fungsi seperti findElm dan findBefore, yang membantu memahami posisi elemen dalam struktur data. Pada modul ini didapatkan juga berhasil menghitung jumlah total nilai semua elemen menggunakan fungsi totalInfo, yang menunjukkan bagaimana linked list dapat digunakan untuk operasi agregasi. Selain itu, pemahaman tentang alokasi dan dealokasi memori menjadi penting dalam manajemen memori saat menggunakan struktur data dinamis ini. Secara keseluruhan, praktikum ini memberikan pemahaman yang lebih mendalam tentang linked list dan aplikasinya dalam pemrograman, membangun dasar yang kuat untuk mempelajari konsep yang lebih kompleks di masa depan.