Telkom University Purwokerto

## Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHubyang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. AsistenPraktikum:4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folderdalam repositoryakan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

| Nomor | Pertemuan | Penamaan |
|---|---|---|
| 1 | Penganalan Bahasa C++ Bagian Pertama | 01_Pengenalan_CPP_Bagian_1 |
| 2 | Pengenalan Bahasa C++ Bagian Kedua | 02_Pengenalan_CPP_Bagian_2 |
| 3 | Abstract Data Type | 03_Abstract_Data_Type |
| 4 | Single Linked List Bagian Pertama | 04_Single_Linked_List_Bagian_1 |
| 5 | Single Linked List Bagian Kedua | 05_Single_Linked_List_Bagian_2 |
| 6 | Double Linked List Bagian Pertama | 06_Double_Linked_List_Bagian_1 |
| 7 | Stack | 07_Stack |
| 8 | Queue | 08_Queue |
| 9 | Assessment Bagian Pertama | 09_Assessment_Bagian_1 |
| 10 | Tree Bagian Pertama | 10_Tree_Bagian_1 |
| 11 | Tree Bagian Kedua | 11_Tree_Bagian_2 |
| 12 | Asistensi Tugas Besar | 12_Asistensi_Tugas_Besar |
| 13 | Multi Linked List | 13_Multi_Linked_List |
| 14 | Graph | 14_Graph |
| 15 | Assessment Bagian Kedua | 15_Assessment_Bagian_2 |
| 16 | Tugas Besar | 16_Tugas_Besar |

5. **Jam Praktikum:**
    - Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwalpraktikumadalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktuini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
        - **07.30 - 0G.30**: Sesi ini mencakup **tutorial, diskusi, dan kasus problem- solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
            - **60 menit pertama**: Tugas terbimbing.
            - **60 menit kedua**: Tugas mandiri.
6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

    **nama_repo/nama_pertemuan/TP_Pertemuan_Ke.md**

    Sebagai contoh:

    **STD_Yudha_Islalmi_Sulistya_XXXXXXXX/01_Running_Modul/TP_01.md**

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

# LAPORAN PRAKTIKUM
# MODUL 5
# SINGLE LINKED LIST (BAGIAN KEDUA)


**Disusun Oleh :**

**Izzaty Zahara Br Barus – 2311104052**

**Kelas :**

**SE-07-02**

**Dosen :**

**Wahyu Andi Saputra, S.pd,M.Eng**

**PROGRAM STUDI SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**

**2024**

Telkom University Purwokerto

# I. 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

# II. LANDASAN TEORI

Searching merupakan operasi dasar list dengan melakukan aktivitas pencarian terhadap node tertentu. Proses ini berjalan dengan mengunjungi setiap node dan berhenti setelah node yang dicari ketemu. Dengan melakukan operasi searching, operasi-operasi seperti insert after, delete after, dan update akan lebih mudah. Semua fungsi dasar diatas merupakan bagian dari ADT dari singgle linked list, dan aplikasi pada bahasa pemrograman Cp semua ADT tersebut tersimpan dalam file *.c dan file *.h

# III. GUIDE

1. Guide1
    a. Syntax

Telkom University Purwokerto

```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 inserlast(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;
    }
}

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 deletElement(Node* head, int x){
    if (head == nullptr){
        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 != 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);

    inserlast(head, tail, 11);
    inserlast(head, tail, 14);
    inserlast(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 likend list" << endl;
    else
        cout << "Elemen ditemukan pada index " << result << endl;

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

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

    return 0;

}
```

b. Penjelasan

Kode ini mengimplementasikan struktur *linked list* dalam C++. Berikut penjelasan singkat dari setiap fungsi:

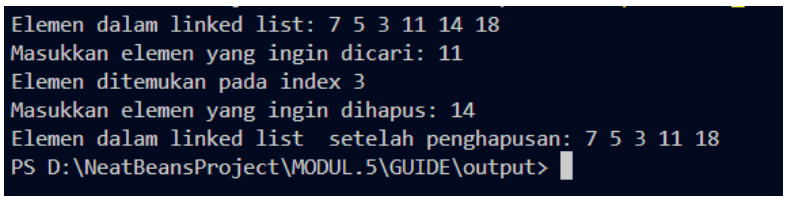
1. Struct Node: Mendefinisikan node dari *linked list* yang memiliki *data* (nilai integer) dan pointer next yang menunjuk ke node berikutnya.
2. insertfirst: Menambahkan elemen baru di awal *linked list*. Jika tail kosong, maka tail juga menunjuk ke node baru tersebut.
3. inserlast: Menambahkan elemen baru di akhir *linked list*. Jika head kosong, head dan tail akan menunjuk ke node baru tersebut.
4. findElement: Mencari elemen dengan nilai tertentu dalam *linked list* dan mengembalikan indeksnya. Jika tidak ditemukan, fungsi mengembalikan -1.
5. display: Menampilkan semua elemen dalam *linked list* secara berurutan.
6. deletElement: Menghapus elemen pertama dengan nilai tertentu dalam *linked list*. Jika elemen tidak ditemukan, tidak ada yang dihapus.
7. main: Fungsi utama yang menginisialisasi *linked list*, menambahkan elemen di awal dan akhir, mencari elemen yang diinginkan, menghapus elemen, dan menampilkan hasil sebelum dan sesudah penghapusan.

Kesalahan kecil:

- Fungsi inserlast seharusnya insertlast.
- Fungsi deletElement seharusnya deleteElement.

Setelah perbaikan, kode ini memungkinkan pengguna menambahkan, mencari, dan menghapus elemen dari *linked list*.

c. Output




## IV. UNGUIDED

### 1. TASK 1

a. Syntax

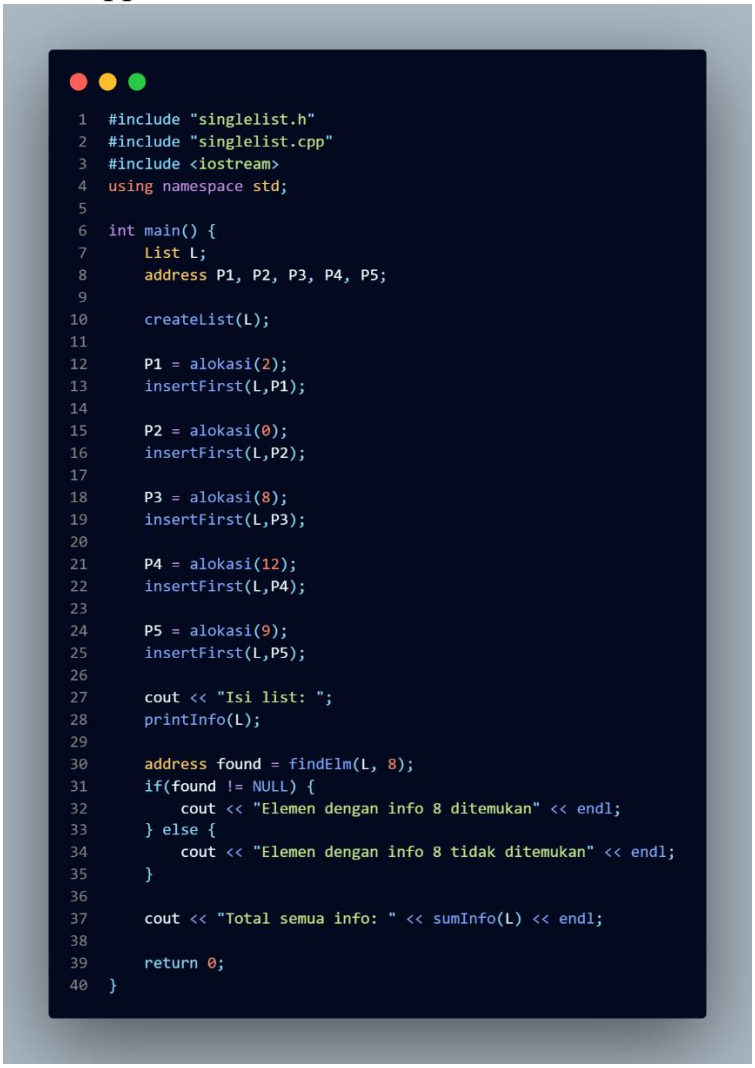
singlelist.h

```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(List L);
void insertFirst(List &L, address P);
address findElm(List L, infotype x);
int sumInfo(List L);

#endif
```

singlelist.cpp


```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(List L);
void insertFirst(List &L, address P);
address findElm(List L, infotype x);
int sumInfo(List L);

#endif
```

main.cpp



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

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);

    cout << "Isi list: ";
    printInfo(L);

    address found = findElm(L, 8);
    if(found != NULL) {
        cout << "Elemen dengan info 8 ditemukan" << endl;
    } else {
        cout << "Elemen dengan info 8 tidak ditemukan" << endl;
    }

    cout << "Total semua info: " << sumInfo(L) << endl;

    return 0;
}
```

b. Penjelasan

1. Header File (singlelist.h):

   - Di sini didefinisikan struktur data untuk elemen list (ElmList) yang memiliki dua bagian: info (tipe integer) dan next (pointer ke elemen berikutnya).

   - Struktur List didefinisikan untuk menyimpan alamat elemen pertama dari list.

   - Fungsi-fungsi yang akan digunakan seperti createList, alokasi, dealokasi, printInfo, insertFirst, findElm, dan sumInfo juga dideklarasikan.

2. Implementasi (singlelist.cpp):

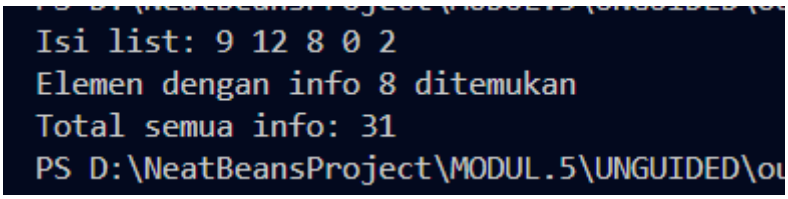
- createList: Membuat list kosong dengan mengatur pointer first menjadi NULL.
- alokasi: Mengalokasikan memori untuk elemen baru dan mengisi nilai info.
- dealokasi: Mengembalikan memori yang digunakan oleh elemen ke sistem.
- printInfo: Menampilkan semua elemen dalam list dengan mencetak nilai info.
- insertFirst: Menambahkan elemen baru di awal list.
- findElm: Mencari elemen dalam list berdasarkan nilai info yang diberikan. Jika ditemukan, mengembalikan alamat elemen tersebut, jika tidak, mengembalikan NULL.
- sumInfo: Menghitung total dari semua nilai info dalam list.

3. Program Utama (main.cpp):

- Membuat list baru dan mengalokasikan beberapa elemen dengan nilai tertentu (2, 0, 8, 12, 9).
- Menambahkan elemen-elemen tersebut ke list di posisi paling depan.
- Menampilkan isi list.
- Mencari elemen dengan nilai 8 dan menampilkan hasil pencarian.
- Menghitung dan menampilkan total dari semua nilai dalam list.

Secara keseluruhan, program ini mengimplementasikan struktur data linked list sederhana yang memungkinkan kita untuk menyimpan, menambah, mencari, dan menjumlahkan elemen-elemen dalam list.

c. Output



# V. KESIMPULAN

Dalam praktikum ini, kita memahami dan mengimplementasikan operasi dasar pada struktur *single linked list*, termasuk operasi *insert* di awal dan akhir, *search*, *delete*, dan penghitungan total elemen dalam list. Struktur linked list memungkinkan penyimpanan dan manipulasi data secara dinamis menggunakan pointer. Penggunaan header file dan pemisahan antara deklarasi dan implementasi membuat kode lebih terstruktur dan modular. Praktikum ini memperkuat pemahaman tentang manipulasi data dalam linked list menggunakan bahasa pemrograman C++.