# Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHub yang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
    a. Asisten Praktikum: AndiniNH
    b. Asisten Praktikum : 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 folder dalam repository akan 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 jadwal praktikum adalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
        - **06.30 - 07.30**: Waktu ini 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.

8. **Struktur Laporan Praktikum**

**1. Cover :**

**LAPORAN PRAKTIKUM**
**Modul 5**
**SINGLE LINKED LIST (BAGIAN KEDUA)**

**Disusun Oleh:**
**KAFKA PUTRA RIYADI - 2311104041**
**Kelas:**
**SE 07-02**

**Dosen :**
**Wahyu Andi Saputra, S.Pd., M.Eng**

**PROGRAM STUDI S1 SOFTWARE ENGINEERING**
**FAKULTAS INFORMATIKA**
**TELKOM UNIVERSITY**
**PURWOKERTO**
**2024**

**2. 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

**3. Landasan Teori**

**Searching** adalah operasi dasar dalam ilmu komputer yang digunakan untuk menemukan elemen tertentu atau sekelompok elemen dalam suatu struktur data. Dalam

C++, terdapat berbagai teknik Searching yang dapat digunakan tergantung pada tipe dan organisasi data.

4. **Guided**

pada guided saya bagi menjadi 2 agar tidak terlalu kecil gambar codingannya dan tidak ngeblur

```cpp
#include <iostream>

using namespace std;

// Struktur untuk node dalam linked list

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

// Fungsi untuk menambahkan elemen baru ke awal linked list 11
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 = 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;
}
```

Telkom University Purwokerto

**Lanjutan dari line 58**

```cpp
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;
    }
}

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;
}
```

## 5. Unguided

1. Codingan dengan file **“main(no.1).cpp”**

```cpp
#include "singlelist(no1).h"
#include "singlelist(no.1).cpp"
#include <iostream>
using namespace std;

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

    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);

    return 0;
}
```

Codingan dengan file **“singlelist(no.1).cpp”**

```cpp
#include <iostream>
#include "singlelist(no1).h"

using namespace std;

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

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

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

void printInfo(const 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;
}
```

Codingan dengan file **“singlelist(no.1).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(const List &L);
void insertFirst(List &L, address P);

#endif
```

**Hasil outputannya:**

```
9 12 8 0 2
PS D:\STRUKTUR DATA P5>
```

**2.** Codingan dengan file **“main(no.2).cpp”**

```cpp
#include <iostream>
#include "singlelist(no.2).h"
#include "singlelist(no.2).cpp"

using namespace std;

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

    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);

    address foundElement = findElm(L, 8);
    if (foundElement != NULL) {
        cout << "8 ditemukan dalam list" << endl;
    } else {
        cout << "8 tidak ditemukan dalam list" << endl;
    }

    return 0;
}
```

Codingan dengan file **“singlelist(no.2).cpp”**

```cpp
#include "singlelist(no.2).h"
#include <iostream>

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;
}

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

int sumInfo(List L) {
    int total = 0;
    address P = L.First;
    while (P != NULL) {
        total += P->info;
        P = P->next;
    }
    return total;
}
```

Codingan dengan file **“singlelist(no.2).h”**

```cpp
#ifndef SINGLELIST_H
#define SINGLELIST_H

#include <iostream>

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
```

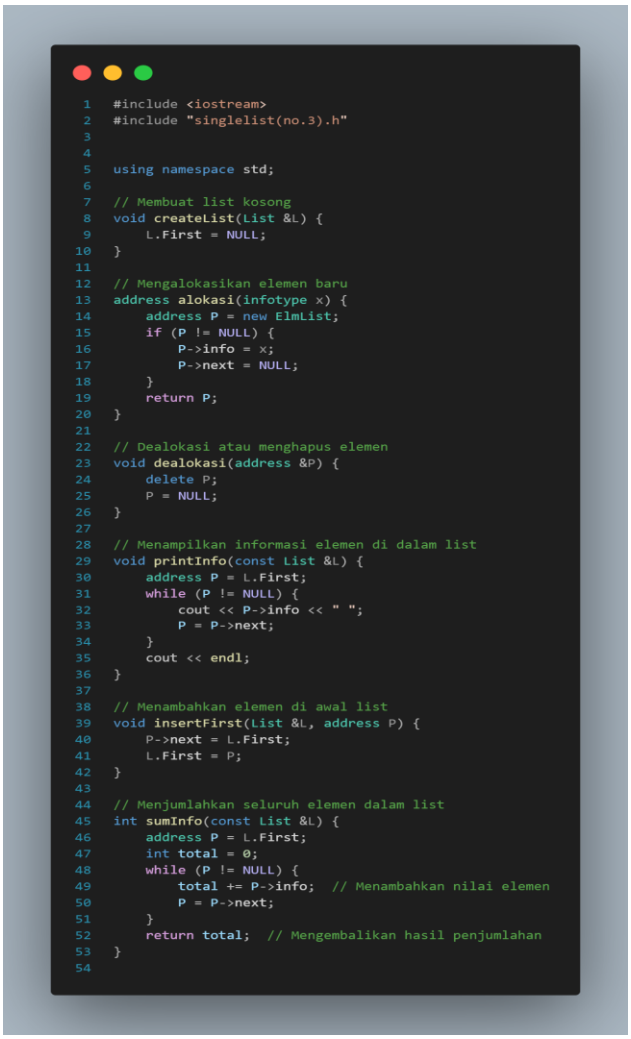
**Outputannya:**

```
8 ditemukan dalam list
PS D:\STRUKTUR DATA P5>
```

3. Codingan dengan file **“main(no.3).cpp”**

```cpp
#include <iostream>
#include "singlelist(no.3).h"
#include "singlelist(no.3).cpp"

using namespace std;

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

    // Membuat list kosong
    createList(L);

    // Menambahkan elemen ke 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);

    // Menampilkan elemen dalam list
    printInfo(L);

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

    return 0;
}
```

Codingan dengan file **“singlelist(no.3).cpp”**



```cpp
#include <iostream>
#include "singlelist(no.3).h"


using namespace std;

// Membuat list kosong
void createList(List &L) {
    L.First = NULL;
}

// Mengalokasikan elemen baru
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != NULL) {
        P->info = x;
        P->next = NULL;
    }
    return P;
}

// Dealokasi atau menghapus elemen
void dealokasi(address &P) {
    delete P;
    P = NULL;
}

// Menampilkan informasi elemen di dalam list
void printInfo(const List &L) {
    address P = L.First;
    while (P != NULL) {
        cout << P->info << " ";
        P = P->next;
    }
    cout << endl;
}

// Menambahkan elemen di awal list
void insertFirst(List &L, address P) {
    P->next = L.First;
    L.First = P;
}

// Menjumlahkan seluruh elemen dalam list
int sumInfo(const List &L) {
    address P = L.First;
    int total = 0;
    while (P != NULL) {
        total += P->info;  // Menambahkan nilai elemen
        P = P->next;
    }
    return total;  // Mengembalikan hasil penjumlahan
}

```

Codingan dengan file **“singlelist(no.3).h**

```cpp
#ifndef SINGLELIST_H
#define SINGLELIST_H

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address First;
};

// Prosedur dan Fungsi ADT
void createList(List &L);
address alokasi(infotype x);
void dealokasi(address &P);
void printInfo(const List &L);
void insertFirst(List &L, address P);
int sumInfo(const List &L);  // Fungsi untuk menjumlahkan elemen-elemen

#endif

```

**Outputannya:**

```
9 12 8 0 2
Total info dari kelima elemen adalah 31
PS D:\STRUKTUR DATA P5>
```

## 6. Kesimpulan

Kesimpulannya yaitu Searching adalah operasi penting dalam ilmu komputer yang digunakan untuk menemukan elemen tertentu dalam struktur data. Dalam C++, terdapat berbagai teknik pencarian yang bisa digunakan sesuai dengan tipe dan organisasi data yang sedang diolah.