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 4
"SINGLE LINKED LIST (BAGIAN PERTAMA)"

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

Linked List adalah salah satu struktur data linear yang terdiri dari serangkaian elemen yang disebut node. Setiap node dalam linked list berisi dua komponen utama:

1. Data: Bagian ini menyimpan informasi atau nilai dari node tersebut.
2. Pointer (Reference): Bagian ini menyimpan alamat atau referensi ke node berikutnya dalam list.

Jenis-jenis Linked List:

1. **Single Linked List:**

   adalah salah satu jenis struktur data **Linked List** di mana setiap node hanya memiliki satu pointer (referensi) yang menunjuk ke node berikutnya. Data pada setiap node disusun secara linear, dan akses terhadap node dilakukan secara sekuensial dari awal hingga akhir. Karena hanya memiliki satu arah aliran, linked list ini disebut "singly" (tunggal).

2. **Double Linked List :**

   adalah jenis struktur data **Linked List** di mana setiap node memiliki dua pointer (atau referensi): satu menunjuk ke node sebelumnya (previous) dan satu lagi menunjuk ke node berikutnya (next)

3. **Circular Linked List** :

   adalah variasi dari struktur data **Linked List** di mana node terakhir terhubung kembali ke node pertama, sehingga membentuk sebuah lingkaran. Dengan kata lain, tidak ada node yang memiliki null sebagai referensi berikutnya (next), dan linked list tidak memiliki akhir.

4.2.1 Pembentukan Komponen-Komponen *List*

Pembentukan komponen list mengacu pada struktur dasar yang digunakan untuk membangun dan mengelola elemen-elemen dalam sebuah list. Dalam konteks Linked List, pembentukan komponen list terdiri dari beberapa elemen penting yang secara umum dapat diterapkan dalam struktur data seperti Singly Linked List, Doubly Linked List, atau Circular Linked List

A. Pembentukan List

Adalah proses pembuatan dan pengaturan elemen-elemen data dalam struktur data **list**. List adalah kumpulan elemen yang dapat diakses secara berurutan, dan bisa menyimpan berbagai jenis data tergantung pada implementasinya.

B. Pengalokasian Memori

Adalah proses reservasi ruang dalam memori komputer yang dilakukan untuk menyimpan data atau objek selama program berjalan.

C. Dealokasi

Adalah proses pelepasan atau pembebasan kembali memori yang sebelumnya telah dialokasikan untuk digunakan oleh suatu program. Ketika memori yang telah dialokasikan tidak lagi dibutuhkan oleh program, memori tersebut harus dikembalikan ke sistem agar bisa digunakan oleh bagian lain dari program atau aplikasi lain yang berjalan pada komputer.

D. Pengecekan List

Adalah proses untuk memeriksa kondisi, status, atau isi dari sebuah list (daftar) dalam konteks struktur data atau pemrograman. Pengecekan list bisa melibatkan berbagai aspek tergantung pada tujuan pemeriksaan.

4.2.2 Insert

A. insert first

Adalah istilah yang digunakan untuk menggambarkan tindakan menambahkan elemen baru di posisi paling depan dalam suatu daftar atau struktur data. Dalam konteks ini, elemen baru akan menjadi yang pertama terlihat atau diakses.

B. insert after

Adalah istilah yang digunakan untuk menggambarkan tindakan menambahkan elemen baru di posisi paling akhir dalam suatu daftar atau struktur data. Dalam konteks ini, elemen baru akan menjadi yang terakhir terlihat atau diakses.

C. insert next

Adalah istilah yang digunakan untuk menambahkan elemen baru setelah elemen tertentu dalam suatu daftar atau struktur data. Dalam konteks ini, elemen baru akan ditempatkan tepat setelah elemen yang sudah ada.

4. **Guided**

Pada code program guided 1 saya bagi menjadi 2 karena jika dijadikan 1 gambar nya keliatan kecil dan nge blur

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

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi List
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
```

Telkom University Purwokerto

Dibawah lanjutan codingan diatas.

```
}
    }

    // Hapus Belakang
    void hapusBelakang() {
        if (!isEmpty()) {
            if (head == tail) {
                delete head;
                head = tail = nullptr; // List menjadi kosong
            } else {
                Node *bantu = head;
                while (bantu->next != tail) {
                    bantu = bantu->next;
                }
                delete tail;
                tail = bantu;
                tail->next = nullptr;
            }
        } else {
            cout << "List kosong!" << endl;
        }
    }

    // Tampilkan List
    void tampil() {
        Node *current = head;
        if (!isEmpty()) {
            while (current != nullptr) {
                cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
                current = current->next;
            }
        } else {
            cout << "List masih kosong!" << endl;
        }
        cout << endl;
    }

    // Hapus List
    void clearList() {
        Node *current = head;
        while (current != nullptr) {
            Node *hapus = current;
            current = current->next;
            delete hapus;
        }
        head = tail = nullptr;
        cout << "List berhasil terhapus!" << endl;
    }

    // Main function
    int main() {
        init();

        // Contoh data mahasiswa
        mahasiswa m1 = {"Alice", "123456"};
        mahasiswa m2 = {"Bob", "654321"};
        mahasiswa m3 = {"Charlie", "112233"};

        // Menambahkan mahasiswa ke dalam list
        insertDepan(m1);
        tampil();
        insertBelakang(m2);
        tampil();
        insertDepan(m3);
        tampil();

        // Menghapus elemen dari list
        hapusDepan();
        tampil();
        hapusBelakang();
        tampil();

        // Menghapus seluruh list
        clearList();

        return 0;
    }
```

Telkom University Purwokerto

Guided 2 juga saya bagi 2 agar tidak blur

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

// Definisi struktur untuk elemen list
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        newNode->next = head;     // Menghubungkan elemen baru ke elemen pertama
        head = newNode;           // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}
```

Telkom University Purwokerto

```cpp
// Menghitung jumlah elemen dalam list
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;           // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count;          // Mengembalikan jumlah elemen
}

// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
}

int main() {
    Node* head = nullptr;  // Membuat list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
    insertLast(head, 30);  // Menambahkan elemen 30 di akhir list

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements(head) << endl;

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

5. Unguided

No.1

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

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

void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data ;
       if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }
    cout << endl;
}


int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List: ";
    printList(head);

    return 0;
}
```

Outputannya:

```
Linked List: 5 -> 10 -> 20
PS D:\unguided struktur data>
```

No.2

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

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

void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void deleteNode(Node** head, int value) {
    Node* temp = *head;
    Node* prev = nullptr;

    if (temp != nullptr && temp->data == value) {
        *head = temp->next;
        delete temp;
        return;
    }

    while (temp != nullptr && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == nullptr) return;

    prev->next = temp->next;

    delete temp;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data ;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }

}


int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List before deletion: ";
    printList(head);

    deleteNode(&head, 10);

    cout << " Linked List after deletion: ";
    printList(head);

    return 0;
}
```

Outputannya:

```
Linked List before deletion: 5 -> 10 -> 20 Linked List after deletion: 5 -> 20
PS D:\unguided struktur data>
```

Telkom University Purwokerto

No.3

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

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

void insertFront(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = *head;
    *head = newNode;
}

void insertBack(Node** head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (*head == nullptr) {
        *head = newNode;
        return;
    }

    Node* temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

bool searchNode(Node* head, int value) {
    Node* temp = head;
    while (temp != nullptr) {
        if (temp->data == value) {
            return true;
        }
        temp = temp->next;
    }
    return false;
}

int countLength(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;
        temp = temp->next;
    }
    return count;
}

void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data ;
        if (temp->next != nullptr) {
            cout << " -> ";
        }
        temp = temp->next;
    }

}

int main() {
    Node* head = nullptr;

    insertFront(&head, 10);
    insertBack(&head, 20);
    insertFront(&head, 5);

    cout << "Linked List: ";
    printList(head);

    int searchValue = 20;
    if (searchNode(head, searchValue)) {
        cout << " Node with value " << searchValue << " found." << endl;
    } else {
        cout << " Node with value " << searchValue << " not found." << endl;
    }

    int length = countLength(head);
    cout << "Length of the linked list: " << length << endl;

    return 0;
}
```

Outputannya:

```
Linked List: 5 -> 10 -> 20 Node with value 20 found.
Length of the linked list: 3
PS D:\unguided struktur data>
```

6. Kesimpulan

Linked List adalah struktur data linear yang terdiri dari serangkaian elemen, atau node, di mana setiap node menyimpan data dan referensi ke node berikutnya. Terdapat tiga jenis utama Linked List:

1. **Single Linked List:** Node memiliki satu pointer yang menunjuk ke node berikutnya, membentuk urutan linear.
2. **Double Linked List:** Node memiliki dua pointer, satu ke node sebelumnya dan satu ke node berikutnya, memungkinkan navigasi dua arah.
3. **Circular Linked List:** Node terakhir terhubung kembali ke node pertama, membentuk struktur melingkar tanpa akhir.

Proses pembentukan komponen list melibatkan pembuatan, pengalokasian memori, dealokasi, dan pengecekan list.

Dalam konteks pengelolaan data dalam Linked List, terdapat beberapa operasi penting seperti:

- **Insert First:** Menambahkan elemen baru di depan list.
- **Insert After:** Menambahkan elemen baru di akhir list.
- **Insert Next:** Menambahkan elemen baru setelah elemen tertentu.

Dengan pemahaman tentang struktur dan operasi Linked List, pengguna dapat mengelola data secara efisien dan fleksibel dalam pemrograman.