LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL IV
SINGLY LINKED LIST

Disusun Oleh :
NAMA : Ridha Akifah
NIM : 103112400132

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Single Linked List (SLL) adalah arsitektur data non-kontigu yang dirancang untuk keluwesan dan skalabilitas. Berbeda dengan penyimpanan memori yang kaku, SLL membangun daftar melalui unit-unit mandiri yang disebut node. Setiap node adalah pasangan antara informasi yang disimpan (datanya) dan sebuah referensi/penunjuk yang berfungsi sebagai "benang penghubung" ke node berikutnya. Akses ke daftar selalu dimulai dari Head, dan rangkaian data ini berakhir ketika penunjuk terakhir bernilai *NULL*. Kekuatan utama SLL terletak pada efisiensi operasi penyisipan dan penghapusan. Cukup dengan mengarahkan ulang penunjuk (pointer) beberapa node, struktur daftar dapat dimodifikasi secara cepat, menjadikannya pilihan unggul untuk aplikasi yang memerlukan manajemen daftar dinamis (seperti antrean atau *playlist*).

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

```
#include <iostream>
#include <cstdlib>
#include "singlylist.h"
#include "singlylist.cpp"

using namespace std;

int main () {
    List L;
    address P;

    CreateList(L);

    cout << "mengisi list menggunakan imterlast..." << endl;

    P = alokasi(9);
```

```
    insertLast (L, P);

    P = alokasi(12);
    insertLast (L, P);

    P = alokasi(8);
    insertLast (L, P);

    P = alokasi(0);
    insertLast (L, P);

    P = alokasi(2);
    insertLast (L, P);

    cout << "ISi List sekarang adalah: ";
    printInfo(L);

    system("pause");
    return 0;

}
```

Guided 2

```
#include "Singlylist.h"

void CreateList(List &L) {
    L.First = Nil;
```

```
}

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

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

void insertFirst (List &L, address P) {
    P->next = L.First;
    L.First = P;
}

void insertLast(List &L, address P ) {
    if (L.First == Nil) {
        // Jika list kosong, insertLast sama dengan insertFirst
        insertFirst(L,P);
    } else {
        // Jika list tidak kosong, cari elemen terakhir
        address Last = L.First;
        while (Last->next != Nil) {
            Last = Last->next;
        }
        // Sambungkan elemen terakhir ke elemen baru (P)
```

```
        Last->next = P;
    }
}

void printInfo(List L) {
    address P = L.First;
    if (P == Nil) {
    std::cout << "List Kosong!" << std::endl;
} else {
    while (P != Nil) {
        std :: cout << P->info << " ";
        P = P->next;
    }
    std :: cout << std :: endl;
}
}
```

Guided 3

```
#ifndef SINGLYLIST_H_INCLUDED
#define SINGLYLIST_H_INCLUDED

#include <iostream>

#define Nil NULL

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 insertFirst(List &L, address P);
void insertLast(List &L, address P);
void printInfo(List L);

#endif
```

Screenshoot Output

```
PS D:\C++\Modul 4> cd "d:\C++\Modul 4\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
mengisi list menggunakan imterlast...
ISi List sekarang adalah: 9 12 8 0 2
Press any key to continue . . .
```

Deskripsi:

Program ini dibuat menggunakan Single Linked List. Program ini dibuat bertujuan untuk mempelajari bagaimana cara menambahkan dan menghampus tanpa bantuan array.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

Playlist.h

```cpp
#ifndef PLAYLIST_H
#define PLAYLIST_H

#include <iostream>
#include <string>

struct Lagu {
    std::string judul;
    std::string penyanyi;
    float durasi;
    Lagu* next;
};

class Playlist {
private:
    Lagu* head;

public:
    Playlist();
    ~Playlist();

    void tambahDepan(const std::string& judul, const std::string& penyanyi, float durasi);
    void tambahBelakang(const std::string& judul, const std::string& penyanyi, float durasi);
    void tambahSetelahKe3(const std::string& judul, const std::string& penyanyi, float durasi);
    void hapusBerdasarkanJudul(const std::string& judul);
    void tampilkanPlaylist() const;
};
```

```
#endif // PLAYLIST_H
```

Playlist.cpp

```
#include "Playlist.h"

Playlist::Playlist() : head(nullptr) {}

Playlist::~Playlist() {
    Lagu* cur = head;
    while (cur != nullptr) {
        Lagu* nxt = cur->next;
        delete cur;
        cur = nxt;
    }
}

void Playlist::tambahDepan(const std::string& judul, const
std::string& penyanyi, float durasi) {
    Lagu* baru = new Lagu{judul, penyanyi, durasi, head};
    head = baru;
       std::cout << "Lagu \"" << judul << "\" ditambahkan di
awal.\n";
}


void Playlist::tambahBelakang(const std::string& judul, const
std::string& penyanyi, float durasi) {
    Lagu* baru = new Lagu{judul, penyanyi, durasi, nullptr};
```

```
    if (head == nullptr) {
        head = baru;
    } else {
        Lagu* temp = head;
        while (temp->next != nullptr) temp = temp->next;
        temp->next = baru;
    }
      std::cout << "Lagu \"" << judul << "\" ditambahkan di
akhir.\n";
}



void Playlist::tambahSetelahKe3(const std::string& judul, const
std::string& penyanyi, float durasi) {
    if (head == nullptr) {

        tambahBelakang(judul, penyanyi, durasi);
        return;
    }

    Lagu* temp = head;
    int count = 1;
    while (temp != nullptr && count < 3) {
        temp = temp->next;
        ++count;
    }

    if (temp == nullptr) {
```

```
        std::cout << "Playlist kurang dari 3 lagu. Menambahkan di
akhir.\n";
        tambahBelakang(judul, penyanyi, durasi);
        return;
    }

    Lagu* baru = new Lagu{judul, penyanyi, durasi, temp->next};
    temp->next = baru;
     std::cout << "Lagu \"" << judul << "\" ditambahkan setelah
lagu ke-3.\n";
}

void Playlist::hapusBerdasarkanJudul(const std::string& judul) {
    if (head == nullptr) {
                std::cout << "Playlist kosong. Tidak ada yang
dihapus.\n";
        return;
    }

    Lagu* cur = head;
    Lagu* prev = nullptr;
    while (cur != nullptr && cur->judul != judul) {
        prev = cur;
        cur = cur->next;
    }

    if (cur == nullptr) {
          std::cout << "Tidak ditemukan lagu dengan judul \"" <<
judul << "\".\n";
        return;
```

```
    }

    if (prev == nullptr) {
        head = cur->next;
    } else {
        prev->next = cur->next;
    }
    delete cur;
    std::cout << "Lagu \"" << judul << "\" berhasil dihapus.\n";
}

void Playlist::tampilkanPlaylist() const {
    if (head == nullptr) {
        std::cout << "Playlist kosong.\n";
        return;
    }
    const Lagu* temp = head;
    int idx = 1;
    std::cout << "=== Playlist ===\n";
    while (temp != nullptr) {
        std::cout << idx++ << ". Judul: " << temp->judul
                  << " | Penyanyi: " << temp->penyanyi
                  << " | Durasi: " << temp->durasi << " menit\n";
        temp = temp->next;
    }
    std::cout << "================\n";
}
```

main2.cpp

```cpp
#include <iostream>
#include <string>
#include <limits>
#include "Playlist.h"
#include "Playlist.cpp"

int main() {
    Playlist myPlaylist;
    int pilihan = -1;

    while (true) {
        std::cout << "\n=== MENU PLAYLIST ===\n";
        std::cout << "1. Tambah lagu di awal\n";
        std::cout << "2. Tambah lagu di akhir\n";
        std::cout << "3. Tambah lagu setelah lagu ke-3\n";
        std::cout << "4. Hapus lagu berdasarkan judul\n";
        std::cout << "5. Tampilkan playlist\n";
        std::cout << "0. Keluar\n";
        std::cout << "Pilih: ";
        if (!(std::cin >> pilihan)) {
            std::cin.clear();
std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
            std::cout << "Input tidak valid. Masukkan angka.\n";
            continue;
        }
std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
```

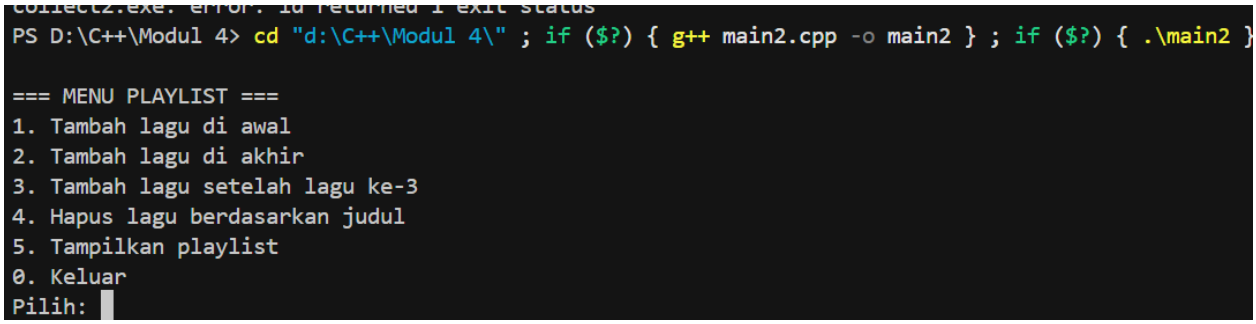
```
        if (pilihan == 0) {
            std::cout << "Keluar dari program.\n";
            break;
        }

        std::string judul, penyanyi;
        float durasi = 0.0f;

        switch (pilihan) {
            case 1:
                std::cout << "Masukkan judul lagu: ";
                std::getline(std::cin, judul);
                std::cout << "Masukkan penyanyi: ";
                std::getline(std::cin, penyanyi);
                    std::cout << "Masukkan durasi (menit, contoh
3.5): ";
                if (!(std::cin >> durasi)) {
                    std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                        std::cout << "Durasi tidak valid. Batalkan
operasi.\n";
                } else {
                            myPlaylist.tambahDepan(judul, penyanyi,
durasi);

std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                }
```

```
                break;


            case 2:
                std::cout << "Masukkan judul lagu: ";
                std::getline(std::cin, judul);
                std::cout << "Masukkan penyanyi: ";
                std::getline(std::cin, penyanyi);
                std::cout << "Masukkan durasi (menit): ";
                if (!(std::cin >> durasi)) {
                    std::cin.clear();
std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                    std::cout << "Durasi tidak valid. Batalkan
operasi.\n";
                } else {
                    myPlaylist.tambahBelakang(judul, penyanyi,
durasi);
std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                }
                break;


            case 3:
                std::cout << "Masukkan judul lagu: ";
                std::getline(std::cin, judul);
                std::cout << "Masukkan penyanyi: ";
                std::getline(std::cin, penyanyi);
                std::cout << "Masukkan durasi (menit): ";
                if (!(std::cin >> durasi)) {
```

```
                    std::cin.clear();

std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                      std::cout << "Durasi tidak valid. Batalkan
operasi.\n";
                } else {
                     myPlaylist.tambahSetelahKe3(judul, penyanyi,
durasi);

std::cin.ignore(std::numeric_limits<std::streamsize>::max(),
'\n');
                }
                break;


            case 4:
                    std::cout << "Masukkan judul lagu yang ingin
dihapus: ";
                std::getline(std::cin, judul);
                myPlaylist.hapusBerdasarkanJudul(judul);
                break;


            case 5:
                myPlaylist.tampilkanPlaylist();
                break;


            default:
                std::cout << "Pilihan tidak valid.\n";
                break;
        }
    }
```

```
    return 0;
}
```

Screenshoot Output



Deskripsi:

Program ini menggunakan struktur data Single Linked List untuk mengelola playlist lagu, di mana setiap lagu disimpan dalam node yang berisi atribut judul, penyanyi, dan durasi. Kelas Playlist menyediakan berbagai operasi seperti menambah lagu di awal, di akhir, dan setelah lagu ke-3, menghapus lagu berdasarkan judul, serta menampilkan seluruh lagu dalam playlist. File Playlist.h mendefinisikan struktur data dan deklarasi fungsi, Playlist.cpp berisi implementasi fungsi-fungsi tersebut, dan main.cpp menyediakan antarmuka menu interaktif agar pengguna dapat menambah, menghapus, atau melihat daftar lagu secara dinamis menggunakan linked list.

D. Kesimpulan

Pada modul ini, telah dipelajari penerapan struktur data Single Linked List dalam mengelola data secara dinamis menggunakan bahasa pemrograman C++. Melalui studi kasus playlist lagu, mahasiswa memahami bagaimana setiap node merepresentasikan satu lagu dengan atribut tertentu (judul, penyanyi, dan durasi), serta bagaimana operasi dasar seperti penambahan node di awal, di akhir, dan di posisi tertentu, penghapusan node berdasarkan kunci (judul lagu), serta penelusuran dan penampilan seluruh data dilakukan menggunakan pointer. Implementasi program dibagi ke dalam tiga file (Playlist.h, Playlist.cpp, dan main.cpp) untuk melatih konsep modular programming dan data encapsulation. Dengan demikian, mahasiswa diharapkan mampu memahami konsep dasar linked list dan dapat mengembangkannya untuk berbagai aplikasi pengelolaan data yang memerlukan struktur dinamis.

E. Referensi

https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170

https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd