LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 4
SINGLE LINKED LIST

Disusun Oleh :

NAMA : RAIHAN DZAKY MUFLIH

NIM : 103112430029

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Single Linked List adalah struktur data dinamis yang terdiri dari rangkaian elemen bernama node di mana setiap node menyimpan data dan satu pointer yang menunjuk ke node berikutnya. Struktur ini memungkinkan penambahan dan penghapusan elemen secara fleksibel tanpa perlu mengatur ulang seluruh data seperti pada array. Elemen pertama disebut head, sedangkan elemen terakhir menunjuk ke nullptr. Operasi dasar yang dapat dilakukan pada single linked list meliputi menambah node di awal, akhir, atau posisi tertentu, menghapus node berdasarkan nilai atau posisi, serta menelusuri seluruh node untuk menampilkan isi data secara berurutan.

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

Guided 1

#singlyList.h

```
#ifndef SINGLYLISH_H_INLCLUDED
#define SINGLYLISH_H_INLCLUDED
#include <iostream>
#define NIL NULL

typedef int infotype;
typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct list {
    address first;
};

// Deklarasi Prosedur dan Fungsi Primitif
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
```

#singlyList.cpp

```cpp
#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, interLast sama dengan insertFirst
        insertFirst(L, P);
    } else {
        // Jika list tidak kosong, cari element terakhir
        address Last = L.first;
        while (Last->next != NIL) {
            Last = Last->next;
        }
        // Sambungkan elemen terakhir dengan elemen terbaru (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;
    }

}
```

#main.cpp

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

using namespace std;

int main()
{
    list L;
    address P; // Cukup satu pointer untuk digunakan berulang kali

    createList(L);

    cout << "Mengisi list menggunaka interLast..." << endl;

    // Mengisi list sesuai urutan
    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;
}
```

Screenshots Output

```
[Running] cd "d:\C++\MODUL 4\" && g++ main.cpp -o main && "d:\C++\MODUL 4\"main
Mengisi list menggunaka interLast...
Isi list sekarang adalah: 9 12 8 0 2
Press any key to continue . . .
```

Deskripsi:
Program di atas merupakan implementasi dasar Single Linked List, yang digunakan untuk menyimpan dan mengelola sekumpulan data secara dinamis. Program ini menyediakan beberapa operasi utama, yaitu membuat list baru, menambah node di awal maupun diakhir, menghapus node dari memori, serta menampilkan seluruh isi list. Dengan struktur ini, data dapat ditambahkan atau dihapus dengan fleksibel tanpa perlu menggeser elemen lain seperti pada array.

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

Unguided 1

#playlist.h

```
//playlist.h
#ifndef PLAYLIST_H
#define PLAYLIST_H

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

struct lagu {
   string judul;
   string penyanyi;
   float durasi;
   lagu* next;

};

class playlist {
private :
    lagu* head;

public :
    playlist();
    ~playlist();
    void tambahAwal(string judul, string penyanyi, float durasi);
    void tambahAkhir(string judul, string penyanyi, float durasi);
    void tambahSetelahPlaylist3(string judul, string penyanyi,
float durasi);
    void hapusLagu(string judul);
    void tampilSeluruhLagu();
};


#endif
```

#playlist.cpp

```cpp
//playlist.cpp
#include "playlist.h"

playlist::playlist() {
    head = nullptr;
}

playlist::~playlist() {
    lagu* temp;
    while (head != nullptr) {
        temp = head;
        head = head->next;
        delete temp;
    }
}

void playlist::tambahAwal(string judul, string penyanyi, float
durasi) {
    lagu* baru = new lagu{judul, penyanyi, durasi, head};
    head = baru;
    cout << "Lagu \"" << judul << "\" berhasil ditambahkan di awal
playlist.\n";
}

void playlist::tambahAkhir(string judul, 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;
    }
    cout << "Lagu \"" << judul << "\" berhasil ditambahkan di
akhir playlist.\n";
}

void playlist::tambahSetelahPlaylist3(string judul, string
```

```
penyanyi, float durasi) {
    lagu* baru = new lagu{judul, penyanyi, durasi, nullptr};
    lagu* temp = head;
    int posisi = 1;

    while (temp != nullptr && posisi < 3) {
        temp = temp->next;
        posisi++;
    }

    if (temp == nullptr) {
        cout << "Playlist kurang dari 3 lagu, tidak bisa menambah
setelah ke-3.\n";
        delete baru;
    } else {
        baru->next = temp->next;
        temp->next = baru;
        cout << "Lagu \"" << judul << "\" berhasil ditambahkan
setelah lagu ke-3.\n";
    }
}

void playlist::hapusLagu(string judul) {
    if (head == nullptr) {
        cout << "Playlist kosong.\n";
        return;
    }

    lagu* temp = head;
    lagu* prev = nullptr;

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

    if (temp == nullptr) {
        cout << "Lagu \"" << judul << "\" tidak ditemukan dalam
playlist.\n";
        return;
    }

    if (prev == nullptr) {
```

```
        head = head->next;
    } else {
        prev->next = temp->next;
    }

    delete temp;
    cout << "Lagu \"" << judul << "\" berhasil dihapus.\n";
}

void playlist::tampilSeluruhLagu() {
    if (head == nullptr) {
        cout << "Playlist kosong.\n";
        return;
    }

    lagu* temp = head;
    int no = 1;

    cout << "\n=== Daftar Lagu dalam Playlist ===\n";
    while (temp != nullptr) {
        cout << no++ << ". Judul: " << temp->judul
             << " | Penyanyi: " << temp->penyanyi
             << " | Durasi: " << temp->durasi << " menit\n";
        temp = temp->next;
    }
    cout << "================================\n";
}
```

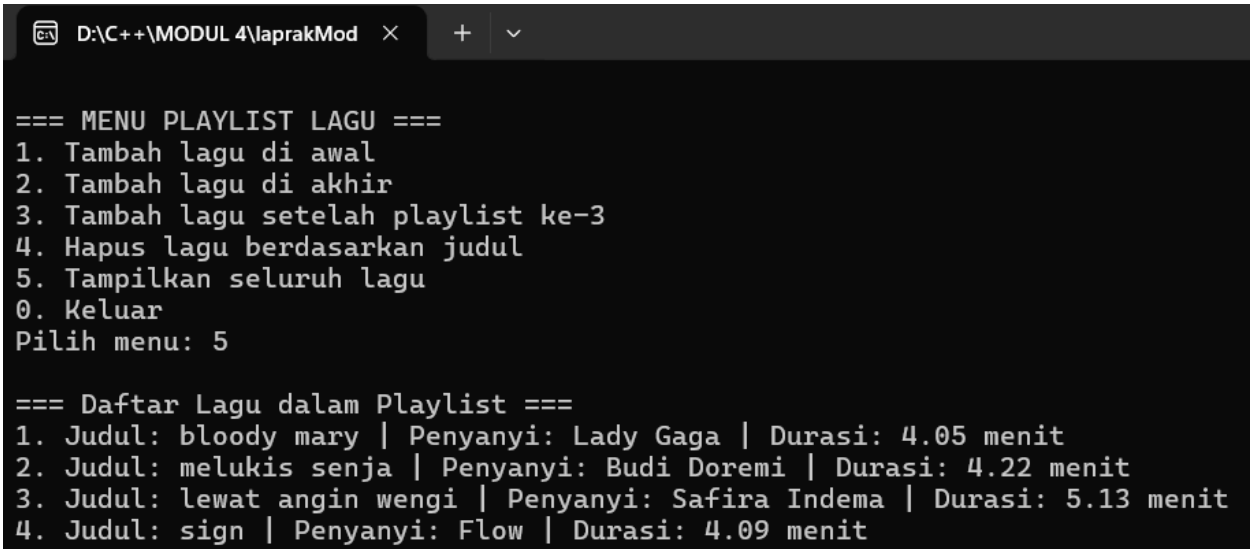
#main.cpp

```
//main.cpp
#include "playlist.h"
#include "playlist.cpp"

int main() {
    playlist daftar;
    int pilihan;
    string judul, penyanyi;
    float durasi;
```

```
    do {
        cout << "\n=== MENU PLAYLIST LAGU ===\n";
        cout << "1. Tambah lagu di awal\n";
        cout << "2. Tambah lagu di akhir\n";
        cout << "3. Tambah lagu setelah playlist ke-3\n";
        cout << "4. Hapus lagu berdasarkan judul\n";
        cout << "5. Tampilkan seluruh lagu\n";
        cout << "0. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore();

        switch (pilihan) {
            case 1:
                cout << "Masukkan judul lagu: ";
                getline(cin, judul);
                cout << "Masukkan nama penyanyi: ";
                getline(cin, penyanyi);
                cout << "Masukkan durasi (menit): ";
                cin >> durasi;
                daftar.tambahAwal(judul, penyanyi, durasi);
                break;

            case 2:
                cout << "Masukkan judul lagu: ";
                getline(cin, judul);
                cout << "Masukkan nama penyanyi: ";
                getline(cin, penyanyi);
                cout << "Masukkan durasi (menit): ";
                cin >> durasi;
                daftar.tambahAkhir(judul, penyanyi, durasi);
                break;

            case 3:
                cout << "Masukkan judul lagu: ";
                getline(cin, judul);
                cout << "Masukkan nama penyanyi: ";
                getline(cin, penyanyi);
                cout << "Masukkan durasi (menit): ";
                cin >> durasi;
                daftar.tambahSetelahPlaylist3(judul, penyanyi,
durasi);
```

```
                break;

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

            case 5:
                daftar.tampilSeluruhLagu();
                break;

            case 0:
                cout << "Keluar dari program...\n";
                break;

            default:
                cout << "Pilihan tidak valid!\n";
        }

    } while (pilihan != 0);

    return 0;
}
```

Screenshots Output



Deskripsi:
Program ini adalah program untuk mengelola data lagu. Setiap lagu disimpan dalam node yang berisi judul, penyanyi, durasi, dan pointer ke lagu berikutnya. Program ini menyediakan berbagai operasi penting, seperti menambah lagu di awal, di akhir, atau setelah lagu ke-3, menghapus lagu berdasarkan judul, serta menampilkan seluruh lagu dalam playlist. Dengan menggunakan single linked list playlist dapat diubah dengan mudah tanpa harus mengatur ulang seluruh data.

D. Kesimpulan

Dari praktikum ini dapat disimpulkan bahwa Single Linked List merupakan struktur data dinamis yang efisien untuk menyimpan dan mengelola data secara fleksibel. Melalui penerapan dapat dipahami cara kerja operasi dasar seperti menambah, menghapus, dan menampilkan data tanpa perlu menggeser elemen lain seperti pada array. Dengan memanfaatkan pointer untuk menghubungkan setiap node linked list memudahkan proses manipulasi data dan menjadi dasar penting dalam pengembangan struktur data yang lebih kompleks.

E. Referensi

Chris Okasaki (1995). "Purely Functional Random-Access Lists". *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*: 86–95. doi:10.1145/224164.224187.

Brodnik, Andrej; Carlsson, Svante; Sedgewick, Robert; Munro, JI; Demaine, ED (1999), *Resizable Arrays in Optimal Time and Space (Technical Report CS-99-09)* (PDF), Department of Computer Science, University of Waterloo

Knuth, Donald (1998). *The Art of Computer Programming*. Vol. 3: *Sorting and Searching* (2nd ed.). Addison-Wesley. p. 547. ISBN 978-0-201-89685-5.