LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL IV
SINGLY LINKED LIST (BAGIAN PERTAMA)

Disusun Oleh :
Muhammad Fathammubina
NIM : 10 3112430188

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Abstract Data Type (ADT)

Tipe data yang didefinisikan bersama dengan sekumpulan operasi (fungsi/prosedur) yang bisa dilakukan terhadapnya. Singkatnya, ADT merupakan data + operasi yang boleh dilakukan dan tidak fokus pada cara penyimpanan data, tapi pada apa yang bisa dilakukan. Komponen dalam ADT meliputi :

1. Type — struktur data (misal struct Mahasiswa)
2. Primitif (operasi dasar), meliputi:

    - Konstruktor (Make...) : membuat objek
    - Selector (Get...) : mengambil nilai komponen
    - Mutator (Set...) : mengubah nilai komponen
    - Validator : memeriksa validitas data
    - Destructor : menghapus atau melepaskan memori
    - Baca/Tulis (I/O) : interaksi dengan input/output
    - Operator relasional / aritmatika
    - Konversi tipe

Struktur file ADT memiliki 3 file utama, yaitu :

- .h = Spesifikasi / header yang berisi struct dan deklarasi fungsi
- .cpp = Implementasi yang berisi fungsi-fungsi dari header
- Main.cpp = Program utama yang memakai ADT

Linked List dan Pointer

Linked List adalah salah satu bentuk struktur data yang fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Struktur ini berupa serangkaian elemen data yang saling berkait (berhubungan). Setiap elemen data bisa berupa data tunggal (satu variabel, misal string nama) atau data majemuk (record yang terdiri dari berbagai tipe data, misal Data Mahasiswa).Implementasi Linked List yang umum menggunakan Pointer. Penggunaan pointer lebih dipilih karena:

- Bersifat dinamis, berbeda dengan Array yang statis.
- Cocok dengan sifat Linked List yang fleksibel dan bergandengan.
- Lebih mudah dalam menangani Linked List dibandingkan Array.

Dalam implementasi Linked List dengan pointer, pengaksesan elemen menggunakan operator -> atau tanda titik ..

Singly Linked List

Singly Linked List adalah model ADT Linked List yang hanya memiliki satu arah pointer. Setiap elemen (disebut juga node atau simpul) merupakan tempat penyimpanan data pada memori tertentu. Elemen terdiri dari dua bagian utama:

- Data (Info): Informasi utama yang tersimpan dalam sebuah elemen.

- Successor (Next): Bagian elemen yang berfungsi sebagai penghubung antar elemen dengan menunjuk alamat elemen di depannya.

Sifat dan Istilah Kunci :

- Satu Pointer: Singly Linked List hanya memerlukan satu buah pointer per node (yaitu next).
- Pembacaan Maju: Hanya dapat melakukan pembacaan secara maju.
- first/head: Pointer pada list yang menunjuk alamat elemen pertama list.
- NULL/Nil: Artinya tidak memiliki nilai, tidak mengacu ke mana pun, atau kosong. Node akhir menunjuk ke Nil (kecuali untuk list sirkular).
- Manipulasi: Relatif mudah saat melakukan penyisipan atau penghapusan di tengah list.

Operasi Dasar ADT (Primitif List)

Operasi-operasi dasar pada Linked List disebut juga operasi primitif.

A. Manajemen Memori

- Penciptaan/Inisialisasi List (createList): Proses membentuk list baru dengan mengeset nilai awal first (dan last, jika ada) dengan Nil.
- Pengalokasian Memori (alokasi): Proses mengalokasikan memori untuk setiap elemen data baru dari heap. Pada C++, syntax yang digunakan adalah new.
- Dealokasi (dealokasi): Proses menghapus dan membebaskan memori alamat yang telah dialokasikan. Pada C++, syntax yang digunakan adalah delete P. Tujuannya adalah mengembalikan memori yang digunakan P ke sistem.

B. Manipulasi Elemen

a. Penyisipan (Insert First/Last/After): Metode memasukkan elemen data baru ke dalam list.

    - Insert First: Memasukkan elemen data di awal list.
    - Insert Last: Memasukkan elemen data di akhir list.
    - Insert After: Memasukkan node baru setelah node tertentu yang ditunjuk.

b. Penghapusan (Delete First/Last/After): Pengambilan atau penghapusan elemen dari list.

    - Delete First: Penghapusan elemen pada awal list.
    - Delete Last: Penghapusan elemen dari akhir list.
    - Delete After: Penghapusan node setelah node tertentu.

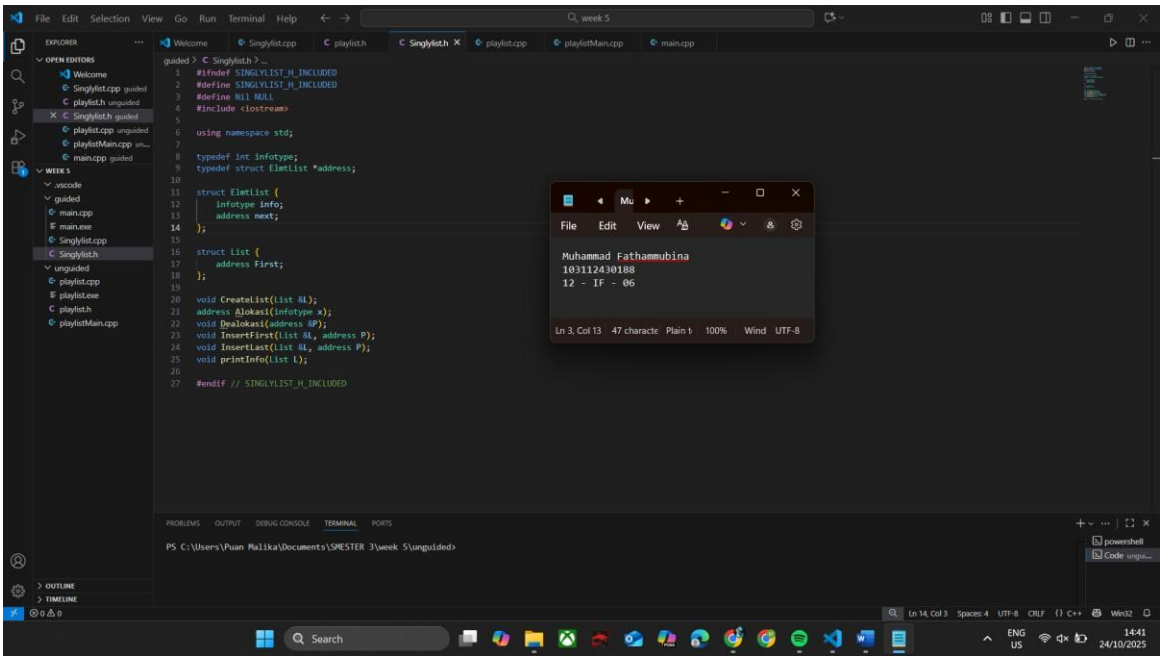
c. Akses Data

- Penelusuran/Tampilan (View/printInfo): Mengunjungi setiap node secara sekuensial dan menampilkan data yang tersimpan pada node tersebut.
- Pencarian (Searching): Operasi untuk menemukan elemen list.
- Pengubahan Isi (Update): Operasi dasar untuk mengubah data yang ada di dalam list, biasanya didahului oleh proses pencarian.

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

Guided 1

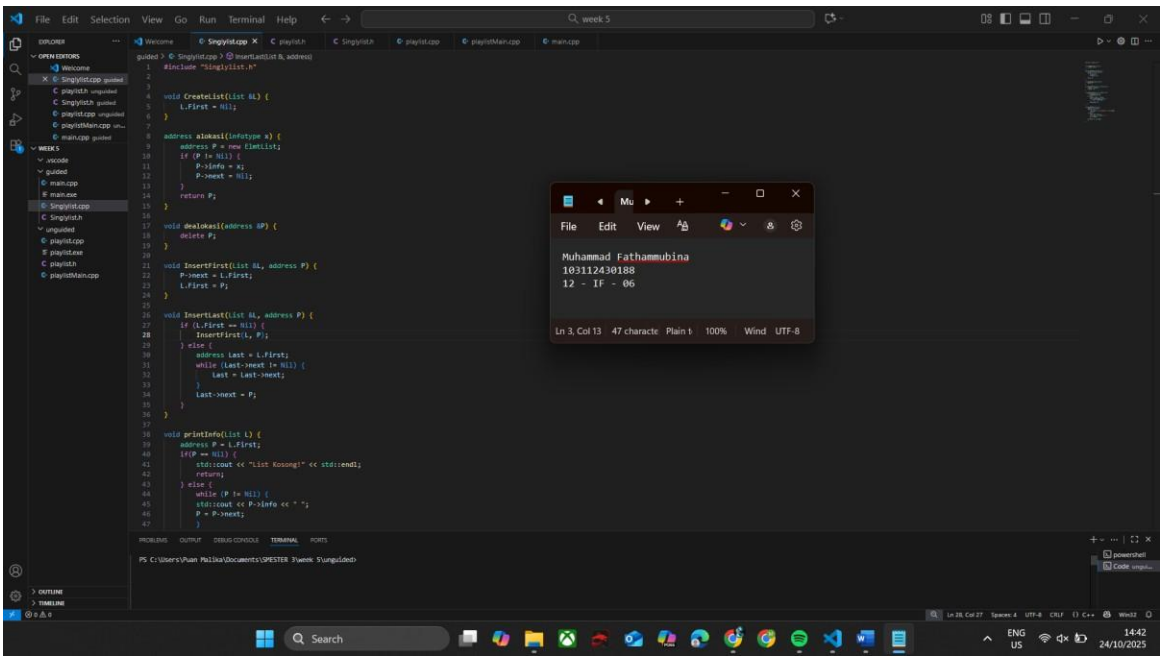
Singlylist.h




```h
#ifndef SINGLYLIST_H_INCLUDED
#define SINGLYLIST_H_INCLUDED
#define Nil NULL
#include <iostream>

using namespace std;

typedef int infotype;
typedef struct ElmtList *address;

struct ElmtList {
    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 // SINGLYLIST_H_INCLUDED
```

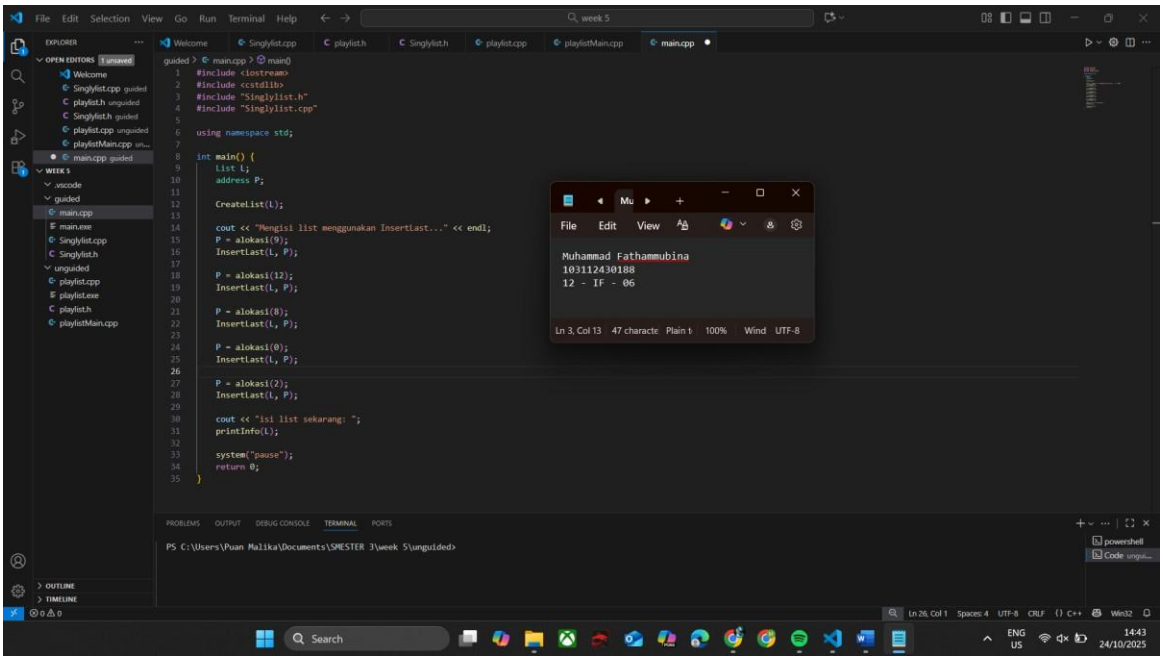
Singlylist.cpp




```
#include "Singlylist.h"


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

address alokasi(infotype x) {
    address P = new ElmtList;
    if (P != Nil) {
        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) {
        InsertFirst(L, P);
    } else {
        address Last = L.First;
        while (Last->next != Nil) {
            Last = Last->next;
        }
        Last->next = P;
    }
}

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

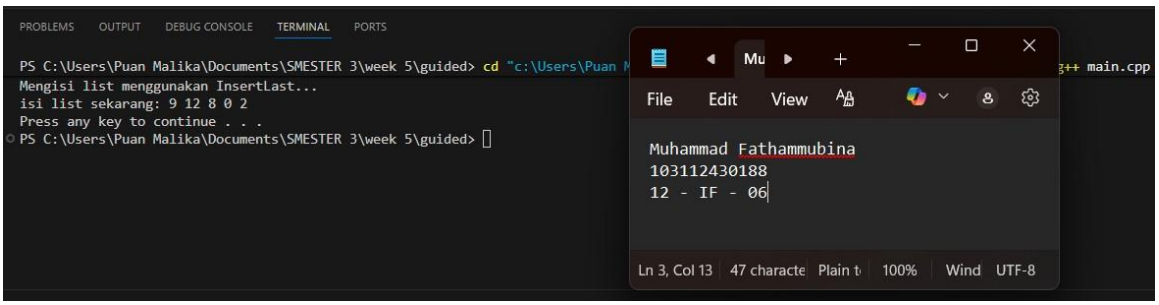
main.cpp




```cpp
#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 InsertLast..." << 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: ";
    printInfo(L);
```

```
    system("pause");
    return 0;
}
```

Screenshots Output

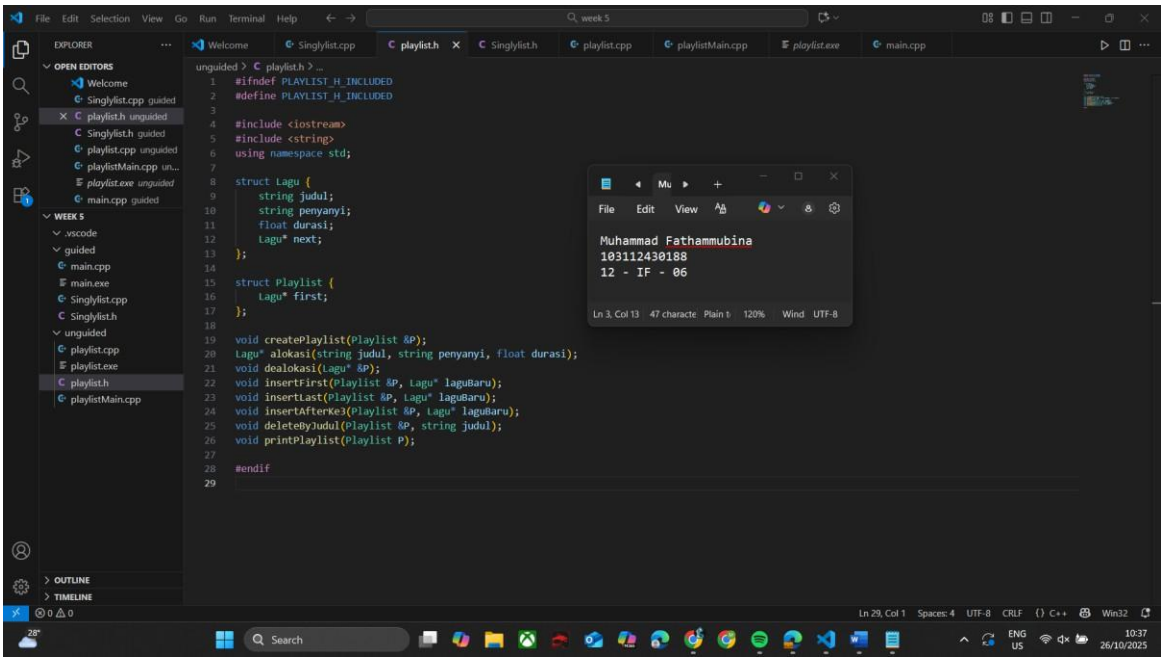


Deskripsi:

Program di atas berfungsi untuk menyimpan dan mengatur data bertipe integer secara dinamis dengan bantuan pointer. Setiap node di dalam list punya dua bagian, yaitu info untuk menyimpan nilai dan next untuk menunjuk ke node berikutnya. Program ini dibagi menjadi tiga file: SinglyList.h, SinglyList.cpp, dan main.cpp. Di file SinglyList.h, terdapat struktur ElmtList dan List, serta deklarasi fungsi-fungsi utama seperti CreateList, Alokasi, Dealokasi, InsertFirst, InsertLast, dan printInfo. Lalu di SinglyList.cpp, semua fungsi tersebut diimplementasikan—mulai dari membuat list kosong, menambah node di awal dan di akhir list, menghapus node dari memori, sampai menampilkan seluruh isi list. Terakhir, di main.cpp, program diuji dengan cara membuat list baru, menambahkan beberapa angka (9, 12, 8, 0, dan 2), lalu menampilkan hasilnya di layar. Hasil output menunjukkan kalau data berhasil disimpan dan ditampilkan secara berurutan.

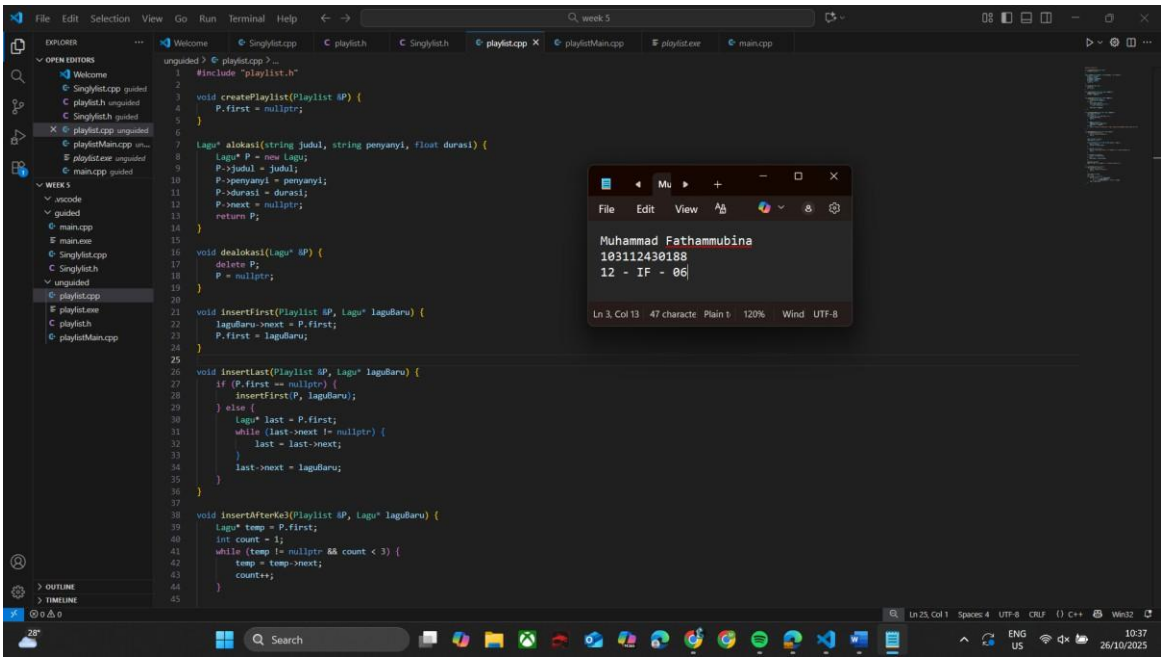
Unguided 1

playlist.h




```
#endif#ifndef PLAYLIST_H_INCLUDED
#define PLAYLIST_H_INCLUDED

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

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

struct Playlist {
    Lagu* first;
};

void createPlaylist(Playlist &P);
Lagu* alokasi(string judul, string penyanyi, float durasi);
void dealokasi(Lagu* &P);
void insertFirst(Playlist &P, Lagu* laguBaru);
void insertLast(Playlist &P, Lagu* laguBaru);
void insertAfterKe3(Playlist &P, Lagu* laguBaru);
void deleteByJudul(Playlist &P, string judul);
void printPlaylist(Playlist P);

#endif
```

playlist.cpp

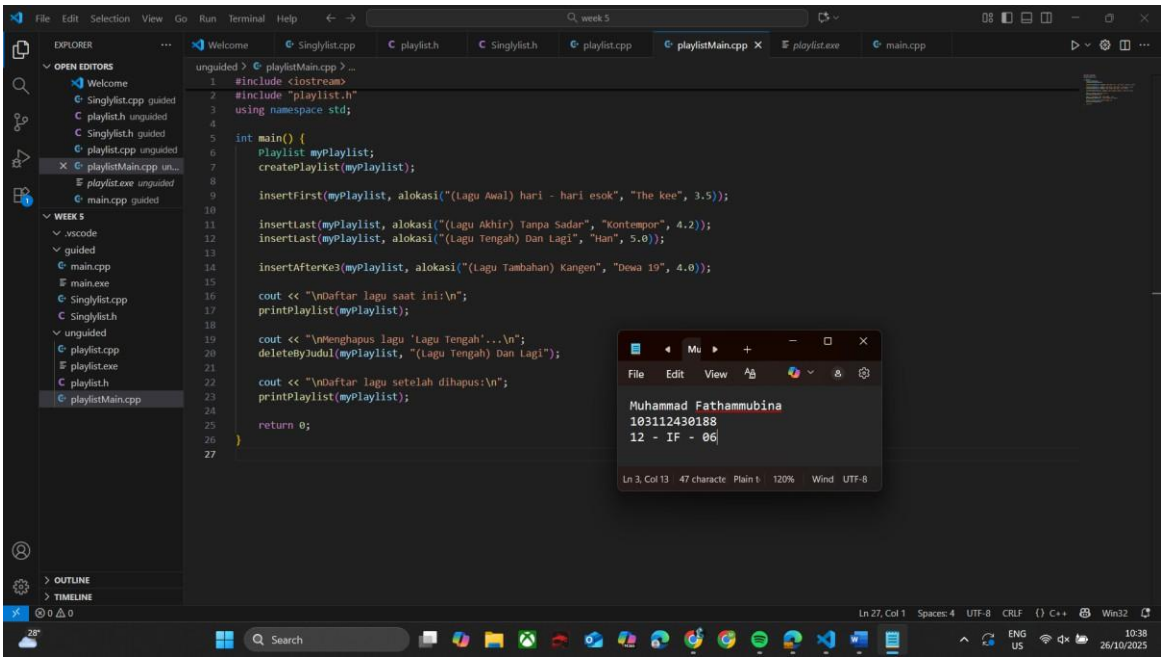



```cpp
#include "playlist.h"

void createPlaylist(Playlist &P) {
    P.first = nullptr;
}

Lagu* alokasi(string judul, string penyanyi, float durasi) {
    Lagu* P = new Lagu;
    P->judul = judul;
    P->penyanyi = penyanyi;
    P->durasi = durasi;
    P->next = nullptr;
    return P;
}

void dealokasi(Lagu* &P) {
    delete P;
    P = nullptr;
}

void insertFirst(Playlist &P, Lagu* laguBaru) {
    laguBaru->next = P.first;
    P.first = laguBaru;
}

void insertLast(Playlist &P, Lagu* laguBaru) {
    if (P.first == nullptr) {
        insertFirst(P, laguBaru);
    } else {
        Lagu* last = P.first;
        while (last->next != nullptr) {
```

```
            last = last->next;
        }
        last->next = laguBaru;
    }
}

void insertAfterKe3(Playlist &P, Lagu* laguBaru) {
    Lagu* temp = P.first;
    int count = 1;
    while (temp != nullptr && count < 3) {
        temp = temp->next;
        count++;
    }

    if (temp != nullptr) {
        laguBaru->next = temp->next;
        temp->next = laguBaru;
    } else {
        cout << "Playlist kurang dari 3 lagu, tidak bisa menambah setelah
lagu ke-3.\n";
    }
}

void deleteByJudul(Playlist &P, string judul) {
    if (P.first == nullptr) {
        cout << "Playlist kosong.\n";
        return;
    }

    Lagu* current = P.first;
    Lagu* prev = nullptr;

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

    if (current == nullptr) {
        cout << "Lagu dengan judul \"" << judul << "\" tidak
ditemukan.\n";
        return;
    }

    if (current == P.first) {
        P.first = current->next;
    } else {
        prev->next = current->next;
    }
```

```
    dealokasi(current);
    cout << "Lagu \"" << judul << "\" berhasil dihapus.\n";
}

void printPlaylist(Playlist P) {
    if (P.first == nullptr) {
        cout << "Playlist kosong.\n";
        return;
    }

    Lagu* temp = P.first;
    int i = 1;
    while (temp != nullptr) {
        cout << i++ << ". " << temp->judul
             << " - " << temp->penyanyi
             << " (" << temp->durasi << " menit)" << endl;
        temp = temp->next;
    }
}
```

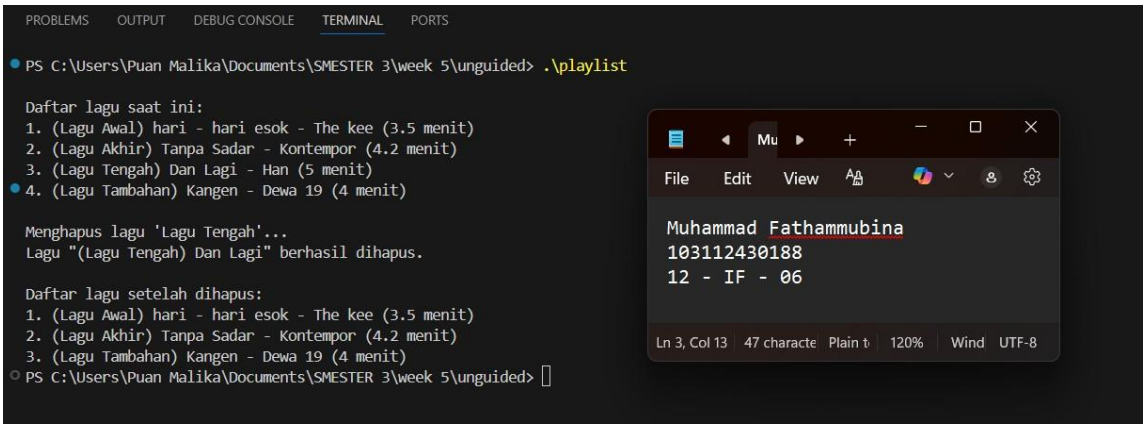
playlistMain.cpp




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

int main() {
    Playlist myPlaylist;
    createPlaylist(myPlaylist);

    insertFirst(myPlaylist, alokasi("(Lagu Awal) hari - hari esok", "The kee", 3.5));

    insertLast(myPlaylist, alokasi("(Lagu Akhir) Tanpa Sadar", "Kontempor", 4.2));
    insertLast(myPlaylist, alokasi("(Lagu Tengah) Dan Lagi", "Han", 5.0));

    insertAfterKe3(myPlaylist, alokasi("(Lagu Tambahan) Kangen", "Dewa 19", 4.0));

    cout << "\nDaftar lagu saat ini:\n";
    printPlaylist(myPlaylist);

    cout << "\nMenghapus lagu 'Lagu Tengah'...\n";
    deleteByJudul(myPlaylist, "(Lagu Tengah) Dan Lagi");

    cout << "\nDaftar lagu setelah dihapus:\n";
    printPlaylist(myPlaylist);

    return 0;
```

```
}
```

Screenshots Output



Deskripsi:

Program diatas mengimplementasikan Abstract Data Type (ADT) Linked List untuk mengelola sebuah Playlist lagu. Struktur dasar terdiri dari struct Lagu sebagai node yang menyimpan data (judul, penyanyi, durasi) dan pointer next yang menghubungkan ke lagu berikutnya, serta struct Playlist yang hanya berisi pointer first untuk menunjuk lagu pertama. Seluruh fungsi operasi Linked List telah tersedia: createPlaylist menginisialisasi playlist kosong; alokasi membuat node lagu baru dari heap menggunakan new; fungsi penyisipan insertFirst, insertLast, dan insertAfterKe3 menangani penambahan node di berbagai posisi; deleteByJudul mencari dan menghapus node tertentu sambil menjaga integritas list; dealokasi mengembalikan memori node ke sistem dengan delete dan mencegah dangling pointer dengan mengatur pointer ke nullptr; dan printPlaylist menampilkan seluruh isi list secara berurutan. Semua fungsi manipulasi list (insert, delete) menggunakan pointer secara ekstensif untuk navigasi dan penugasan alamat, menjadikan program ini contoh lengkap dari implementasi Single Linked List dinamis.

C. Kesimpulan

Program yang disajikan merupakan implementasi dari Abstract Data Type (ADT) Singly Linked List untuk mengelola daftar lagu (Playlist) secara dinamis, di mana setiap lagu adalah sebuah node yang dihubungkan melalui pointer next. Struktur data ini bersifat fleksibel, dapat tumbuh dan mengerut sesuai kebutuhan, dan menggunakan pointer karena sifatnya yang dinamis dibandingkan array yang statis. Operasi-operasi dasar Linked List atau primitif list yang disediakan meliputi createPlaylist untuk inisialisasi list kosong, alokasi menggunakan syntax new untuk membuat node baru, dealokasi menggunakan delete dan pass by reference (Lagu* &P) untuk membebaskan memori dengan aman, serta fungsi manipulasi data seperti insertFirst, insertLast, insertAfterKe3 untuk penyisipan, deleteByJudul untuk penghapusan, dan printPlaylist untuk penelusuran (View) seluruh isi list. Inti dari implementasi ini adalah penggunaan pointer untuk

navigasi, penugasan alamat, dan dereferensi, yang memungkinkan semua elemen list saling berkait dan dapat diakses hanya dari pointer first.

D. Referensi

Muliono, R. (2017). Abstract Data Type (ADT). Universitas Multimedia Nusantara.

Chrismono, A., Handayani,D, UN. (2022) METODE SINGLE LINKED LIST PEMILIHAN SMA / SMK DISEMARANG BERDASARKAN ALGORITMA NEAREST NEIGHBOR DAN ANALISIS SPASIAL BUFFERING. JEEE-U (Journal of Electrical and Electronic Engineering-UMSIDA) 6(1):70-8 DOI:10.21070/jeeeu.v6i1.1634