

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 4 & 5
ABSTRACT DATA TYPE (ADT)**



Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Linked List adalah salah satu bentuk struktur data yang berupa serangkaian elemen data yang saling berkait dan bersifat fleksibel karena dapat tumbuh dan mengerut sesuai kebutuhan. Linked List dapat diimplementasikan menggunakan Array atau Pointer, namun pointer lebih sering digunakan karena Linked List bersifat dinamis, dan sifatnya yang fleksibel lebih cocok dengan pointer. Dalam implementasinya dengan pointer, pengaksesan elemen dapat menggunakan operator panah (->).

Salah satu model ADT Linked List yang dipelajari adalah Singly Linked List.

Singly Linked List hanya memiliki satu arah pointer dan node akhirnya menunjuk ke NULL/Nil. Setiap elemen (Node/Simpul) terdiri dari dua bagian: Data/Info dan Successor/Next, yang berfungsi sebagai penghubung ke elemen di depannya.

Operasi-operasi dasar pada Linked List, yang merupakan bagian dari Abstract Data Type (ADT), meliputi Penciptaan List (CreateList), Manajemen Memori (alokasi dan dealokasi), Pengecekan List (ListEmpty), Penyisipan (InsertFirst, InsertLast, InsertAfter), Penghapusan (delFirst, delLast, delAfter, delp), Penelusuran/Penampilan (printInfo), Pencarian (Searching/findElm), dan Pengubahan isi elemen (Update).

Operasi Searching adalah proses pencarian terhadap node tertentu yang berjalan dengan mengunjungi setiap node dan berhenti setelah node yang dicari ketemu; operasi ini penting untuk mempermudah operasi seperti insert after, delete after, dan update. Semua fungsi dasar tersebut dikenal sebagai operasi primitif dan disimpan dalam file prototipe (.h) dan implementasi (.cpp atau .c).

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

“Singlylist.h”

```
#ifndef LIST_H_INCLUDED
#define LIST_H_INCLUDED
#include <iostream>
#define Nil NULL
using namespace std;

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
```

“Singlylist.cpp”

```
#include "singlylist.h"

// Membuat sebuah list baru yang masih kosong
void createList(list &L){
    L.first = Nil;
}

// Mengalokasikan memori untuk sebuah elemen baru
// dan mengisi elemen tersebut dengan data (X)
address alokasi(infotype X){
    address P = new elmList;
    P->info = X;
    P->next = Nil;
    return P;
}

// Menghapus atau membebaskan memori dari sebuah elemen
void dealokasi(address &P){
    delete P;
}

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

// Menambahkan sebuah elemen baru di akhir list
void insertLast(list &L, address P){
    if(L.first == Nil){
        // Jika list masih kosong, elemen baru menjadi elemen pertama
        insertFirst(L, P);
    } else {
        // Jika list sudah ada isinya, cari elemen terakhir
        address Last = L.first;
        while (Last->next != Nil){
            Last = Last->next;
        }
    }
}
```

```

    }
    // Sambahkan elemen terakhir dengan elemen baru
    Last->next = P;
}
}

// Menampilkan semua isi data dari setiap elemen di list
void printInfo(list L){
    address P = L.first;
    if (P == Nil) {
        cout << "List kosong" << endl;
    } else {
        while (P != Nil) {
            cout << P->info << " ";
            P = P->next;
        }
        cout << endl;
    }
}
}

```

“main.cpp”

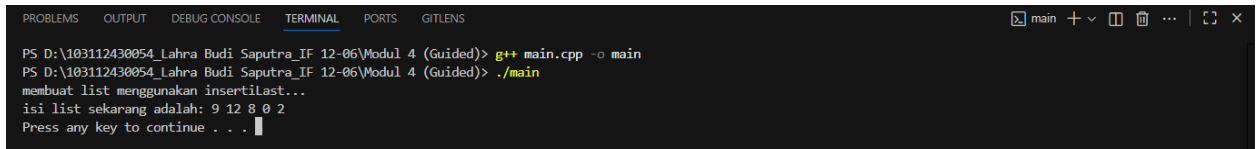
```

#include <iostream>
#include <cstdlib>
#include "singlylist.h"
#include "singlylist.cpp"
using namespace std;

int main() {
    list L;
    address P;
    createList(L);
    cout << "membuat list menggunakan insertiLast..."<<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;
}

```

Screenshots Output



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 4 (Guided)> g++ main.cpp -o main
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 4 (Guided)> ./main
membuat list menggunakan insertLast...
isi list sekarang adalah: 9 12 8 0 2
Press any key to continue . . .
```

Deskripsi:

Ketiga file tersebut mengimplementasikan operasi dasar pada struktur data Singly Linked List menggunakan C++.

File "**Singlylist.h**" berfungsi sebagai header yang mendeklarasikan struktur data dasar, termasuk `elmlist` (node dengan info dan next pointer) dan `list` (dengan first pointer), serta prototipe enam fungsi primitif: `createList`, `alokasi`, `dealokasi`, `insertFirst`, `insertLast`, dan `printInfo`.

File "**Singlylist.cpp**" menyediakan implementasi dari keenam fungsi tersebut, di mana `createList` menginisialisasi list menjadi kosong, `alokasi` membuat node baru, `insertFirst` menyisipkan node di awal, `insertLast` menyisipkan node dengan menelusuri list hingga akhir, dan `printInfo` menampilkan semua data dalam urutan maju.

Terakhir, file "**main.cpp**" adalah program utama yang menguji fungsi-fungsi tersebut dengan membuat list kosong, kemudian menyisipkan lima elemen data secara berurutan (9, 12, 8, 0, 2) menggunakan `insertLast`, dan menampilkan hasil akhir list.

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

Unguided 2

(Playlist.h)

```
#ifndef PLAYLIST_H_INCLUDED
#define PLAYLIST_H_INCLUDED

#include <iostream>
#include <string>

#define Nil NULL
using namespace std;

// Struktur Node Lagu
struct Lagu {
    string judul;
    string penyanyi;
    float durasi;
```

```

};

typedef Lagu infotype;
typedef struct elmlist *address;

struct elmlist {
    infotype info;
    address next;
};

struct list {
    address first;
};

// Prototipe Fungsi
void createList(list &L);
address alokasi(infotype X);
void dealokasi(address &P);
address findElm(list L, string judulLagu);

void insertFirst(list &L, address P);
void insertLast(list &L, address P);
void insertAfter(list &L, address P, address Prec);
void delP(list &L, string judulLagu);
void printInfo(list L);

#endif

```

(Playlist.cpp)

```

#include "Playlist.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) {
    if (P != Nil) {
        delete P;
        P = Nil;
    }
}

```

```

// Mencari elemen berdasarkan judul
address findElm(list L, string judulLagu) {
    address P = L.first;
    while (P != Nil) {
        if (P->info.judul == judulLagu) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

// Tambah lagu di awal playlist
void insertFirst(list &L, address P) {
    P->next = L.first;
    L.first = P;
}

// Tambah lagu di akhir playlist
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 insertAfter(list &L, address P, address Prec) {
    if (Prec != Nil) {
        P->next = Prec->next;
        Prec->next = P;
    }
}

// Hapus lagu berdasarkan judul
void delP(list &L, string judulLagu) {
    address P = L.first;
    address Prec = Nil;

    // Cari elemen yang akan dihapus
    while (P != Nil && P->info.judul != judulLagu) {
        Prec = P;
        P = P->next;
    }

    if (P == Nil) {

```

```

        cout << "Lagu \"" << judulLagu << "\" tidak ditemukan." << endl;
        return;
    }

    // Proses penghapusan
    address DeletedNode = P;
    if (P == L.first) {
        // Hapus First
        L.first = P->next;
    } else {
        // Hapus di tengah/akhir
        Prec->next = P->next;
    }

    // P harus diputus dari rantai sebelum dealokasi
    DeletedNode->next = Nil;
    dealokasi(DeletedNode);
    cout << "Lagu \"" << judulLagu << "\" berhasil dihapus." << endl;
}

// Tampilkan seluruh lagu dalam playlist
void printInfo(list L) {
    address P = L.first;
    if (P == Nil) {
        cout << "Playlist kosong." << endl;
    } else {
        cout << "\n=> Playlist\n";
        int counter = 1;
        while (P != Nil) {
            cout << counter++ << ". " << P->info.judul
                << " (" << P->info.penyanyi << ", " << P->info.durasi << " menit)" <<
endl;
            P = P->next;
        }
    }
}

```

(main.cpp)

```

#include "Playlist.h"
#include "Playlist.cpp"

// Fungsi bantu untuk mendapatkan alamat node ke-N
address getAddressNth(list L, int N) {
    address P = L.first;
    int count = 1;
    while (P != Nil && count < N) {
        P = P->next;
        count++;
    }
}

```



```

    return P;
}

int main() {
    list PlaylistSaya;
    createList(PlaylistSaya);
    address P;

    // Data Lagu Awal
    Lagu L1 = {"Mimpi", "Isyana Sarasvati", 4.3f};
    Lagu L2 = {"Monokrom", "Tulus", 3.4f};
    Lagu L3 = {"Raja", "Dewa 19", 4.0f};
    Lagu L4 = {"Laskar Pelangi", "Nidji", 4.8f};

    // Tambah Lagu di Awal
    P = alokasi(L1);
    insertFirst(PlaylistSaya, P);
    P = alokasi(L2);
    insertFirst(PlaylistSaya, P);

    // Tambah Lagu di Akhir
    P = alokasi(L3);
    insertLast(PlaylistSaya, P);

    cout << "====> PLAYLIST AWAL <=====";
    printInfo(PlaylistSaya);

    // Tambah Lagu Setelah Posisi Ke-3
    P = alokasi(L4);
    int posisiInsert = 3;
    address Prec = getAddressNth(PlaylistSaya, posisiInsert);

    if (Prec != Nil) {
        insertAfter(PlaylistSaya, P, Prec);
        cout << "\nOperasi: Tambah " << L4.judul << " setelah lagu ke-" <<
posisiInsert << "\n";
    } else {
        cout << "\nOperasi: Posisi ke-" << posisiInsert << " tidak ditemukan.\n";
    }

    // Tampilkan Seluruh Lagu
    cout << "=====> PLAYLIST SETELAH INSERT AFTER <=====";
    printInfo(PlaylistSaya);
    // Urutan: [Monokrom, Mimpi, Raja, Laskar Pelangi]

    // Hapus Lagu Berdasarkan Judul
    string judulHapus1 = "Monokrom";
    cout << "\nOperasi: Hapus lagu \"" << judulHapus1 << "\"\n";
    delP(PlaylistSaya, judulHapus1);

```

```

string judulHapus2 = "Raja";
cout << "Operasi: Hapus lagu \"" << judulHapus2 << "\"\n";
delP(PlaylistSaya, judulHapus2);

cout << "\n====> PLAYLIST AKHIR <=====";
printInfo(PlaylistSaya);
// Urutan Akhir: [Mimpi, Laskar Pelangi]

return 0;
}

```

Screenshots Output:

```

PS D:\103112430054_Lahra Budi Saputra_IF 12-06> cd "d:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 4 & 5 (Unguided)" ; if ($?) { g++ main.cpp -o main } ; if ($?) {
.\main }
====> PLAYLIST AWAL <====
=> Playlist
1. Monokrom (Tulus, 3.4 menit)
2. Mimpi (Isyana Sarasvati, 4.3 menit)
3. Raja (Dewa 19, 4 menit)

Operasi: Tambah Laskar Pelangi setelah lagu ke-3
====> PLAYLIST SETELAH INSERT AFTER <====
=> Playlist
1. Monokrom (Tulus, 3.4 menit)
2. Mimpi (Isyana Sarasvati, 4.3 menit)
3. Raja (Dewa 19, 4 menit)
4. Laskar Pelangi (Nidji, 4.8 menit)

Operasi: Hapus lagu "Monokrom"
Lagu "Monokrom" berhasil dihapus.
Operasi: Hapus lagu "Raja"
Lagu "Raja" berhasil dihapus.

====> PLAYLIST AKHIR <====
=> Playlist
1. Mimpi (Isyana Sarasvati, 4.3 menit)
2. Laskar Pelangi (Nidji, 4.8 menit)
PS D:\103112430054_Lahra Budi Saputra_IF 12-06\Modul 4 & 5 (Unguided)>

```

Deskripsi:

Ketiga file tersebut secara keseluruhan mengimplementasikan struktur data Singly Linked List untuk mengelola playlist lagu di C++.

File "Playlist.h" mendeklarasikan ADT, menetapkan struct Lagu sebagai infotype dengan atribut judul, penyanyi, dan durasi, serta mendeklarasikan semua fungsi primitif yang diperlukan seperti insertFirst, insertLast, insertAfter, dan delP. Implementasi dari fungsi-fungsi ini berada di **"Playlist.cpp"**, yang menyediakan logika untuk operasi dasar Linked List, termasuk mekanisme penelusuran (printInfo), alokasi/dealokasi memori, pencarian elemen berdasarkan judul (findElm), dan penanganan kasus penghapusan elemen di awal, tengah, atau akhir list dalam fungsi delP.

File "main.cpp" berfungsi sebagai driver program, mendemonstrasikan kemampuan playlist dengan melakukan serangkaian operasi: menambahkan lagu di awal dan akhir, menyisipkan lagu baru setelah elemen tertentu (lagu ke-3), menampilkan status playlist, dan akhirnya menghapus lagu berdasarkan judul, membuktikan bahwa operasi manipulasi Singly Linked List berfungsi sesuai harapan program.

D. Kesimpulan

Kesimpulannya, program ini sukses mengaplikasikan konsep Singly Linked List dari modul sebagai Abstract Data Type (ADT) untuk manajemen playlist yang dinamis. Pembagian kode menjadi header (.h), implementasi (.cpp), dan main program

menunjukkan praktik pemrograman modular yang baik. Dengan memanfaatkan pointer next, playlist dapat tumbuh dan mengerut secara fleksibel, memungkinkan penambahan lagu di awal (insertFirst), di akhir (insertLast), di tengah (insertAfter), dan penghapusan elemen spesifik (delP) hanya dengan memanipulasi ulang tautan antar node, tanpa perlu menggeser data seperti pada struktur Array.

E. Referensi

- [1] Parlante, N. (n.d.). Linked List Basics. Stanford University CS Education Library.
- [2] Sara, M. R. A. (2020). An Efficient Dynamic Array with Linked List Structure. Indonesian Journal of Computing. Telkom University.