

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL IV & V
SINGLY LINKED LIST**



Disusun oleh :

Junadil Muqorobin (103112400281)

Dosen

Fahrudin Mukti Wibowo S.Kom., M.Eng

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Struktur data adalah suatu cara penyusunan dan pengorganisasian data di dalam komputer agar dapat diakses dan dimanipulasi secara efisien. Dalam konteks pemrograman, struktur data yang baik akan sangat menentukan kinerja program, baik dalam segi waktu maupun penggunaan memori (Anaraindyah, 2024; Wijoyo et al., 2024). Salah satu struktur data yang sering digunakan adalah *singly linked list*, atau dalam Bahasa Indonesia sering disebut senarai berantai tunggal.

Linked list merupakan himpunan simpul (node) yang setiap simpulnya menyimpan informasi data serta pointer (atau referensi) yang menghubungkan ke simpul berikutnya dalam daftar. Dengan demikian, simpul pertama disebut *head*, dan simpul terakhir memiliki pointer yang menunjuk ke *nullptr* atau NULL sebagai penanda akhir list (Anwar et al., 2024). Karena alokasi memori dilakukan secara dinamis—yaitu saat runtime—linked list memungkinkan penambahan dan penghapusan elemen tanpa harus memindahkan elemen lainnya seperti pada array konvensional (Wijoyo et al., 2024; Anaraindyah, 2024).

Pada versi tunggal (*singly*), setiap node hanya memiliki satu arah tautan ke node berikutnya. Struktur ini memudahkan operasi penyisipan di awal atau di tengah list dengan hanya mengubah pointer, namun memiliki kelemahan seperti tidak dapat mengakses elemen secara acak dengan cepat dan hanya dapat dilintasi dari *head* ke *tail* (Anwar et al., 2024). Selain itu, dibandingkan dengan array, struktur linked list menawarkan fleksibilitas ukuran tetapi akses ke elemen individu memerlukan penelusuran berurutan, sehingga dalam skenario tertentu kinerja akses bisa lebih lambat (Wijoyo et al., 2024).

Dalam praktik pengelolaan daftar lagu (playlist), model linked list sangat cocok diterapkan karena playlist bersifat dinamis—dapat ditambah, dihapus, atau disisipkan lagu kapan saja tanpa harus mengetahui jumlah maksimal di awal. Penerapan linked list untuk playlist mencakup node yang berisi atribut judul lagu, penyanyi, dan durasi; kemudian operasi seperti menambah di awal, di akhir, setelah posisi ke-3, menghapus berdasarkan judul, dan menampilkan seluruh daftar adalah representasi nyata dari manipulasi list dinamis tersebut.

B. Guided

1. Implementasi Singly Linked List

Source code singlyList.h

```
#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;
};

// Deklarasi prosedur
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
```

Source code singlyList.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)
    {
        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;
    } else {
```

```

        while (P != Nil) {
            std::cout << P->info << " ";
            P = P->next;
        }
        std::cout << std::endl;
    }
}

```

Source code main.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 adalah: ";
    printInfo(L);

    system("pause");
    return 0;
}

```



Screenshoot Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS I:\ATELYU\semester_3\struktur_data_ptk> cd 'I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\output'
PS I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\output> & .\main.exe
Mengisi list menggunakan insertLast...
Isi list sekarang adalah: 9 12 8 0 2
Press any key to continue . . .
PS I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\output> █
```

Deskripsi:

Program diatas adalah implementasi singly linked list, setiap node hanya memiliki satu arah hubungan ke elemen selanjutnya. Program berjalan dengan langkah kerja sebagai berikut:

- Source code singlyList.h
File singlyList.h digunakan sebagai header file, yaitu mendefinisikan struktur data, tipe data, serta deklarasi fungsi yang digunakan pada program.
 - #define Nil NULL merupakan definisi konstanta Nil sebagai alias dari NULL, yaitu nilai kosong yang menunjukkan pointer.
 - Typedef int infotype; menyatakan tipe data informasi yang disimpan pada setiap node adalah integer.
 - typedef struct ElmList *address; merupakan deklarasi tipe data address sebagai pointer yang mnunjuk ke struct ElmList.
 - Struct ElmList yaitu struct utama untuk menampung semua node dalam list, dan satu pointer First yang menunjuk ke elemen pertama dari linked list.

- Terdapat enam deklarasi fungsi yaitu `CreateList(List &L)` untuk Membuat list kosong. Alokasi (infotype x) untuk mengalokasikan node baru. dealokasi (address &P) untuk Menghapus node dari memori. `insertFirst(List &L, address P)` untuk menyisipkan node di awal list. `insertLast(List &L, address P)` untuk menyisipkan node di akhir list. `printInfo(List L)` untuk Menampilkan seluruh isi list.
- Source code `singlyList.cpp`
 File `singlyList.cpp` merupakan file implementasi dari semua fungsi yang ada di file header.
 - void `CreateList(List &L)` yaitu inialisasi list agar kosong dengan cara mengatur pointer First menjadi Nil.
 - Address alokasi (infotype x) digunakan untuk membuat node baru di memori dengan operator `new`, kemudian node yang dibuat diisi dengan nilai x dan next diatur ke Nil.
 - Void `dealokasi(address &P)` untuk menghapus node yang sudah tidak digunakan dari memori menggunakan `delete`.
 - void `insertFirst(List &L, address P)` digunakan untuk menambahkan node P ke awal list.

- void insertLast(List &L, address P) digunakan untuk menambahkan node P di akhir list.
- Void printInfo(List L) bertugas menampilkan seluruh data dari node secara beruntun mulai dari elemen pertama hingga terakhir, jika list kosong maka akan menampilkan pesan "LIST KOSONG!". Jika tidak maka semua nilai info akan dicetak dengan spasi sebagai pemisah.
- Source code main.cpp
 File main.cpp merupakan program utama yang berfungsi untuk menguji semua fungsi yang telah dibuat pada modul singly linked list
 - Pertama impor file #include "singlyList.h" dan #include "singlyList.cpp" agar fungsi-fungsi list dapat digunakan di program utama.
 - Kemudian deklarasi variabel L sebagai struktur list, dan P sebagai pointer node
 - CreateList(L) menginisialisasi linked list agar dalam keadaan kosong sebelum digunakan.
 - Kemudian program menambahkan elemen (9,12,8,0,2) ke dalam list dengan menggunakan fungsi insertLast(L, P) setelah melakukan alokasi node:
 P = alokasi(9);
 insertLast(L, P);

setiap pemanggilan alokasi() membuat node baru dan insertLast() menempatkannya di urutan terakhir list.

- Kemudian menampilkan seluruh elemen list secara berurutan dengan printInfo(L) sehingga hasilnya adalah:
9 12 8 0 2
- Kemudian ada tambahan system("pause") berfungsi menahan tampilan agar tidak langsung tertutup setelah program selesai dijalankan.

C. Unguided

1. Membuat playlist lagu dengan SLL

Source code playlist.h

```
#ifndef PLAYLIST_H_INCLUDED
#define PLAYLIST_H_INCLUDED

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

#define Nil NULL

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

typedef struct Node *address;

struct Node {
    Lagu info;
```

```

        address next;
    };

    struct Playlist {
        address first;
    };

    // PROTOTYPE FUNGSI
    void createList(Playlist &L);
    address alokasi(Lagu x);
    void dealokasi(address P);
    bool isEmpty(Playlist L);
    void insertFirst(Playlist &L, address P);
    void insertLast(Playlist &L, address P);
    void insertAfterK3(Playlist &L, address P);
    void deleteByJudul(Playlist &L, string judul);
    void printInfo(Playlist L);
#endif

```

Source code playlist.cpp

```

#include "Playlist.h"

void createList(Playlist &L) {
    L.first = Nil;
}

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

void dealokasi(address P) {
    delete P;
}

bool isEmpty(Playlist L) {
    return (L.first == Nil);
}

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

```

```

void insertLast(Playlist &L, address P) {
    if (isEmpty(L)) {
        insertFirst(L, P);
    } else {
        address Q = L.first;
        while (Q->next != Nil) {
            Q = Q->next;
        }
        Q->next = P;
    }
}

void insertAfterK3(Playlist &L, address P) {
    if (isEmpty(L)) {
        cout << "Playlist masih kosong.\n";
        return;
    }
    address Q = L.first;
    int count = 1;

    while (Q != Nil && count < 3) {
        Q = Q->next;
        count++;
    }

    if (Q == Nil) {
        cout << "Kurang dari 3 lagu, menambahkan di akhir.\n";
        insertLast(L, P);
    } else {
        P->next = Q->next;
        Q->next = P;
    }
}

void deleteByJudul(Playlist &L, string judul) {
    if (isEmpty(L)) {
        cout << "Playlist kosong.\n";
        return;
    }

    address P = L.first;
    address prev = Nil;

    while (P != Nil && P->info.judul != judul) {

```

```

        prev = P;
        P = P->next;
    }

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

    if (prev == Nil) {
        L.first = P->next;
    } else {
        prev->next = P->next;
    }

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

void printInfo(Playlist L) {
    if (isEmpty(L)) {
        cout << "Playlist kosong.\n";
        return;
    }

    address P = L.first;
    int i = 1;
    cout << "\n=== Daftar Lagu dalam Playlist ===\n";
    while (P != Nil) {
        cout << i++ << ". Judul: " << P->info.judul
            << " | Penyanyi: " << P->info.penyanyi
            << " | Durasi: " << P->info.durasi << " menit\n";
        P = P->next;
    }
    cout << "=====\n";
}

```

Source code main.cpp

```

#include "Playlist.h"

int main() {
    Playlist L;
    createList(L);
}

```

```
Lagu lagu1 = {"December", "Neck Deep", 3.4};
Lagu lagu2 = {"Sinarengan", "Denny Caknan", 6.5};
Lagu lagu3 = {"Rumah Ke Rumah", "Hindia", 5.1};
Lagu lagu4 = {"Lantas", "Juicy Luicy", 4.0};
Lagu lagu5 = {"Satu-Satu", "Tulus", 4.3};

insertFirst(L, alokasi(lagu1));
insertLast(L, alokasi(lagu2));
insertLast(L, alokasi(lagu3));

printInfo(L);

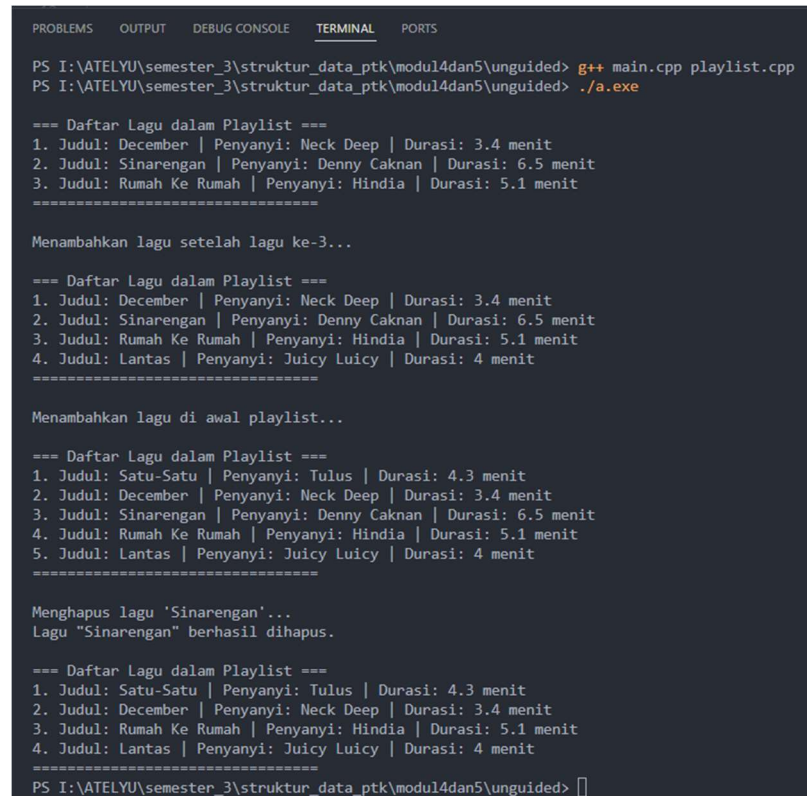
cout << "\nMenambahkan lagu setelah lagu ke-3...\n";
insertAfterK3(L, alokasi(lagu4));
printInfo(L);

cout << "\nMenambahkan lagu di awal playlist...\n";
insertFirst(L, alokasi(lagu5));
printInfo(L);

cout << "\nMenghapus lagu 'Sinarengan'...\n";
deleteByJudul(L, "Sinarengan");
printInfo(L);

return 0;
}
```

Screenshot Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\unguided> g++ main.cpp playlist.cpp
PS I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\unguided> ./a.exe

=== Daftar Lagu dalam Playlist ===
1. Judul: December | Penyanyi: Neck Deep | Durasi: 3.4 menit
2. Judul: Sinarengan | Penyanyi: Denny Caknan | Durasi: 6.5 menit
3. Judul: Rumah Ke Rumah | Penyanyi: Hindia | Durasi: 5.1 menit
=====

Menambahkan lagu setelah lagu ke-3...

=== Daftar Lagu dalam Playlist ===
1. Judul: December | Penyanyi: Neck Deep | Durasi: 3.4 menit
2. Judul: Sinarengan | Penyanyi: Denny Caknan | Durasi: 6.5 menit
3. Judul: Rumah Ke Rumah | Penyanyi: Hindia | Durasi: 5.1 menit
4. Judul: Lantas | Penyanyi: Juicy Luicy | Durasi: 4 menit
=====

Menambahkan lagu di awal playlist...

=== Daftar Lagu dalam Playlist ===
1. Judul: Satu-Satu | Penyanyi: Tulus | Durasi: 4.3 menit
2. Judul: December | Penyanyi: Neck Deep | Durasi: 3.4 menit
3. Judul: Sinarengan | Penyanyi: Denny Caknan | Durasi: 6.5 menit
4. Judul: Rumah Ke Rumah | Penyanyi: Hindia | Durasi: 5.1 menit
5. Judul: Lantas | Penyanyi: Juicy Luicy | Durasi: 4 menit
=====

Menghapus lagu 'Sinarengan'...
Lagu "Sinarengan" berhasil dihapus.

=== Daftar Lagu dalam Playlist ===
1. Judul: Satu-Satu | Penyanyi: Tulus | Durasi: 4.3 menit
2. Judul: December | Penyanyi: Neck Deep | Durasi: 3.4 menit
3. Judul: Rumah Ke Rumah | Penyanyi: Hindia | Durasi: 5.1 menit
4. Judul: Lantas | Penyanyi: Juicy Luicy | Durasi: 4 menit
=====

PS I:\ATELYU\semester_3\struktur_data_ptk\modul4dan5\unguided> 
```

Deskripsi:

Program diatas adalah impelementasi dari single linked list yang digunakan untuk melihat playlist, menambah lagu di list pertama, menambah lagu di list ketiga, dan menghapus lagu tertentu. Program berjalan dengan Langkah kerja sebagai berikut:

- Source code playlist.h
File ini merupakan header file yang berisi deklarasi struktur data dan prototype fungsi sebagai kerangka dasar.
 - Struct lagu berisi struktur yang menggambarkan data lagu yang disimpan pada setiap node list yaitu ada judul, penyanyi dan durasi.
 - typedef struct Node *address; digunakan untuk mendefinisikan tipe

data address sebagai pointer yang merujuk pada node.

- Struct node merupakan elemen dari linked list yang menyimpan info dengan tipe lagu, dan next yaitu pointer yang menunjuk node berikutnya.
- Struct playlist merupakan struktur utama list yang hanya memiliki satu pointer first untuk menunjuk ke node pertama dalam playlist, jika first == Nil, berarti playlist masih kosong.
- Deklarasi fungsi prototype createList untuk membuat list kosong, alokasi untuk membuat node baru, dealokasi untuk menghapus node dari memori, isEmpty untuk memeriksa apakah list kosong, insertFirst, insertLast, insertAfterK3 untuk operasi penambahan lagu, deleteByJudul untuk menghapus lagu berdasarkan judul, dan printInfo untuk menampilkan seluruh isi playlist.
- Source code playlist.cpp
File ini berisi implementasi dari fungsi prototype yang ada di file playlist.h. Berikut adalah langkah kerja dari fungsi yang ada.
 - Pertama createList(Playlist &L) menginisialisasi list dengan mengatur L.first = Nil, yaitu playlist kosong.
 - Kemudian alokasi(Lagu x) membuat node baru di memori menggunakan operator new.
 - Dealokasi(address P) menghapus node dari memori menggunakan delete P.
 - Fungsi isEmpty(Playlist L) mengecek apakah first bernilai Nil. Jika iya playlist dianggap kosong.
 - Kemudian insertFirst(Playlist &L, address P) menambahkan node baru di awal list dengan langkah P->next =

L.first yaitu node baru menunjuk ke node lama, lalu L.first = P yaitu node baru menjadi elemen pertama.

- insertLast(Playlist &L, address P) berfungsi menambahkan node baru di bagian akhir list.
- Kemudian insertAfterK3(Playlist &L, address P) yaitu menyisipkan node baru setelah node ke-3. Jika list kosong maka akan tampil “Playlist masih kosong”. Kemudian menelusuri node satu persatu sambil menghitung posisi $\text{count} < 3$. Jika jumlah node kurang dari 3 \rightarrow tampilkan pesan dan tambahkan di akhir list. Namun jika jumlah node ≥ 3 \rightarrow sambungkan node baru setelah node ke-3.
- Fungsi deleteByJudul(Playlist &L, string judul) digunakan untuk menghapus lagu dengan judul tertentu
- Kemudian printInfo(Playlist L) untuk menampilkan seluruh isi playlist, jika kosong maka tampil pesan “Playlist kosong.” Jika ada maka menampilkan seluruh lagu dalam playlist.

- Source code main.cpp

File main.cpp merupakan program utama yang memanfaatkan semua fungsi-fungsi pada playlist.cpp, berikut adalah langkah kerjanya.

- Pertama akan memerintah membuat playlist kosong dengan Playlist L; createList(L); agar L.first = Nil.
- Kemudian membuat 5 lagu dengan data judul, penyanyi, dan durasi lagu dalam menit.
- Menambah lagu ke playlist dengan insertFirst(L, alokasi(lagu1)); yaitu menambah lagu "December" di awal list. insertLast(L, alokasi(lagu2)); dan

- insertLast(L, alokasi(lagu3)); untuk menambah lagu berikutnya di akhir list.
- Untuk menampilkan playlist memanggil fungsi printInfo(L).
- Kemudian menambah lagu setelah lagu ke-3 menggunakan insertAfterK3(L, alokasi(lagu4));
- Menambah lagu di awal playlist dengan insertFirst(L, alokasi(lagu5)); untuk menambah “Satu-Satu” di posisi pertama.
- Menghapus berdasarkan judul deleteByJudul(L, "Sinarengan"); untuk menghapus node dengan judul tersebut. Dan list setelah di hapus adalah Satu-Satu → December → Rumah Ke Rumah → Lantas.
- Setelah selesai program menampilkan daftar lagu paling update dengan printInfo(L).

D. Kesimpulan

Dari praktikum yang telah dilakukan, dapat disimpulkan bahwa konsep *Singly Linked List* merupakan salah satu bentuk struktur data dinamis yang sangat berguna dalam pengelolaan data yang berubah-ubah ukurannya, seperti pada kasus pembuatan playlist lagu. Berbeda dengan array yang bersifat statis dan memiliki keterbatasan dalam penambahan atau penghapusan elemen, *linked list* memungkinkan proses tersebut dilakukan dengan lebih fleksibel karena setiap data disimpan dalam node yang terhubung melalui pointer. Dengan demikian, pengelolaan data menjadi lebih efisien tanpa perlu memindahkan elemen lain di memori.

Dalam implementasinya, setiap node dalam playlist terdiri atas tiga atribut utama, yaitu judul lagu, penyanyi, dan durasi.

Node-node ini kemudian saling terhubung satu arah membentuk daftar lagu yang dapat diakses secara berurutan dari awal hingga akhir. Melalui fungsi-fungsi yang dibuat pada program, berbagai operasi dasar dapat dilakukan, seperti menambahkan lagu di awal maupun di akhir playlist, menyisipkan lagu setelah elemen tertentu, menghapus lagu berdasarkan judul, serta menampilkan seluruh isi playlist. Setiap operasi tersebut menunjukkan bagaimana pointer bekerja dalam menjaga hubungan antar-node agar struktur list tetap konsisten.

E. Referensi

Anaraindyah, R. (2024, Agustus 26). Struktur data: Pengertian, fungsi, dan penerapannya. *Terapan-TI Vokasi UNESA*. <https://terapanti.vokasi.unesa.ac.id/post/struktur-data-pengertian-fungsi-dan-penerapannya>

Anwar, K., Santiari, N. P. L., Setyowibowo, S., Sigar, T. R., Atho'illah, I., Setyantoro, D., & Emang Smrti, N. N. (2024). *Pengantar struktur data* (Edisi 1). PT Mifandi Mandiri Digital.

Wijoyo, A., Azzahra, A., Nabila, D., Silviana, F., & Lusiyaniti. (2024). Perbandingan struktur linked list dan array dalam manajemen memori. *JRIIN : Jurnal Riset Informatika dan Inovasi*, 1(12), 1290–1293. <https://jurnalmahasiswa.com/index.php/jriin/article/view/957>