

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL VI
DOUBLY 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

Abstraksi Data atau *Abstract Data Type* (ADT) merupakan konsep dalam ilmu komputer yang menjelaskan bagaimana sebuah tipe data didefinisikan berdasarkan perilaku dan operasi yang dapat dilakukan terhadapnya, bukan berdasarkan implementasi internalnya. Dengan kata lain, ADT berfokus pada *apa yang dilakukan*, bukan *bagaimana cara melakukannya*. Salah satu implementasi populer dari ADT adalah *linked list*. *Linked list* merupakan struktur data dinamis yang terdiri atas serangkaian node, di mana setiap node berisi dua bagian utama, yaitu data dan pointer yang mengarah ke node berikutnya. Berbeda dengan array yang menggunakan alokasi memori statis dan bersebelahan, *linked list* memiliki keunggulan karena dapat menambah atau menghapus elemen tanpa perlu menggeser elemen lain di memori (Mbejo, 2025). Struktur ini sangat efisien untuk program yang membutuhkan perubahan ukuran data secara dinamis.

Dari berbagai jenis *linked list*, *Doubly Linked List* (DLL) adalah bentuk yang lebih kompleks dan fleksibel dibanding *Singly Linked List*. Pada DLL, setiap node memiliki dua pointer, yaitu *next* untuk menunjuk node berikutnya dan *prev* untuk menunjuk node sebelumnya. Dengan dua arah penunjuk ini, traversal dapat dilakukan maju maupun mundur, sehingga mempermudah operasi seperti penghapusan node di tengah list tanpa perlu menelusuri list dari awal (Agung, 2024). Meskipun demikian, DLL membutuhkan memori lebih besar karena menyimpan dua pointer pada setiap node, namun biaya tambahan ini sebanding dengan peningkatan efisiensi dalam operasi tertentu seperti *delete* dan *insert* di tengah list (Winarsih, 2022).

Dalam praktik implementasi ADT Doubly Linked List, terdapat beberapa operasi dasar yang perlu dipahami. Prosedur `CreateList` digunakan untuk menginisialisasi list kosong, sedangkan fungsi alokasi dan dealokasi berfungsi untuk membuat dan membebaskan memori node. Operasi `insertLast` menyisipkan elemen baru di bagian akhir list, dan `printInfo` menampilkan seluruh isi data list. Fungsi `findElm` digunakan untuk mencari node tertentu berdasarkan kunci tertentu seperti nomor polisi kendaraan. Selain itu, terdapat pula prosedur penghapusan seperti `deleteFirst`, `deleteLast`, dan `deleteAfter`, yang masing-masing bertugas menghapus node pertama, terakhir, atau node setelah elemen tertentu (Agung, 2024).

Dari segi kompleksitas, operasi penyisipan dan penghapusan pada posisi yang sudah diketahui umumnya memiliki kompleksitas waktu $O(1)$ karena hanya melibatkan perubahan pointer antar node. Namun, jika memerlukan pencarian terlebih dahulu, seperti pada fungsi `findElm`, maka kompleksitasnya menjadi $O(n)$. Sementara itu, dari sisi memori, Doubly Linked List memiliki *overhead* tambahan akibat penyimpanan pointer `prev` di setiap node, namun memberikan kemudahan navigasi dua arah yang sangat bermanfaat dalam berbagai aplikasi (Winarsih, 2022). Dengan fleksibilitasnya yang tinggi, struktur ini memberikan keseimbangan antara efisiensi, kemudahan pengelolaan memori, dan kemudahan navigasi data, sehingga menjadi salah satu topik penting dalam pembelajaran struktur data dan algoritma.

B. Guided

1. Implementasi Doubly Linked List

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

struct Node
{
    int data;
    Node *prev;
    Node *next;
};

Node *ptr_first = NULL;
Node *ptr_last = NULL;

void add_first(int value)
{
    Node *newNode = new Node{value, NULL, ptr_first};
    if(ptr_first == NULL)
    {
        ptr_last = newNode;
    }
    else
    {
        ptr_first->prev = newNode;
    }
    ptr_first = newNode;
}

void add_last(int value)
{
    Node *newNode = new Node{value, ptr_last, NULL};

    if(ptr_last == NULL)
    {
        ptr_first = newNode;
    }
    else
    {
        ptr_last->next = newNode;
    }
    ptr_last = newNode;
}
```

```

void add_target(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        if (current == ptr_last)
        {
            add_last(newValue);
        }
        else
        {
            Node *newNode = new Node{newValue, current, current-
>next};
            current->next->prev = newNode;
            current->next = newNode;
        }
    }
}

void view()
{
    Node *current = ptr_first;
    if (current == NULL)
    {
        cout << "List kosong\n";
        return;
    }
    while (current != NULL)
    {
        cout << current->data << (current->next != NULL ? " <-> "
: "");
        current = current->next;
    }
    cout << endl;
}

void delete_first()
{
    if (ptr_first == NULL)
        return;
}

```

```

    Node *temp = ptr_first;

    if (ptr_first == ptr_last)
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_first = ptr_first->next;
        ptr_first->prev = NULL;
    }
    delete temp;
}

void delete_last()
{
    if (ptr_last == NULL)
        return;

    Node *temp = ptr_last;

    if (ptr_first == ptr_last) // CEKKKK
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_last = ptr_last->prev;
        ptr_last->next = NULL;
    }
    delete temp;
}

void delete_target(int targetValue){
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {

```

```

        if (current == ptr_first)
        {
            delete_first();
            return;
        }
        if (current == ptr_last)
        {
            delete_last();
            return;
        }

        current->prev->next = current->next;
        current->next->prev = current->prev;
        delete current;
    }
}

void edit_node(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while (current != NULL && current->data != targetValue)
    {
        current = current->next;
    }

    if (current != NULL)
    {
        current->data = newValue;
    }
}

int main()
{
    add_first(10);
    add_first(5);
    add_last(20);
    cout << "Awal\t\t\t: ";
    view();

    delete_first();
    cout << "Setelah delete_first\t: ";
    view();
    delete_last();
    cout << "Setelah delete_last\t: ";

```

```


view();

add_last(30);
add_last(40);
cout << "Setelah tambah\t\t: ";
view();

delete_target(30);
cout << "Setelah delete_target\t: ";
view();
}

```

Screenshoot Output:



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS I:\ATELYU\semester_3\struktur_data_ptk> cd 'i:\ATELYU\semester_3\struktur_data_ptk\modul6\output'
PS I:\ATELYU\semester_3\struktur_data_ptk\modul6\output> & .\main.exe
Awal          : 5 <-> 10 <-> 20
Setelah delete_first : 10 <-> 20
Setelah delete_last  : 10
Setelah tambah      : 10 <-> 30 <-> 40
Setelah delete_target : 10 <-> 40
PS I:\ATELYU\semester_3\struktur_data_ptk\modul6\output> 

```

Deskripsi:

Program diatas adalah implementasi doubly linked list, program ini menyimpan kumpulan node yang saling terhubung dua arah (maju dan mundur) melalui pointer next dan prev. Program berjalan dengan langkah kerja sebagai berikut:

- Struktur utama

Struktur yang pertama yaitu struct node yang terdiri dari data untuk menyimpan nilai integer, prev untuk menunjuk ke node sebelumnya, dan next untuk menunjuk ke node berikutnya. Selain itu juga terdapat pointer global Node *ptr_first = NULL; untuk menunjuk node pertama (head) dan Node *ptr_last = NULL; untuk menunjuk node terakhir (tail).

- Fungsi `add_first()`

Fungsi `add_first()` berfungsi untuk menambahkan sebuah node baru di bagian awal (head) dari list. Langkah kerjanya dimulai dengan membuat node baru menggunakan operator `new` yang berisi nilai data yang diberikan, kemudian pointer `next` dari node baru diarahkan ke node pertama sebelumnya (`ptr_first`).

- Fungsi `add_last()`

Fungsi `add_last()` berfungsi menambahkan node baru di bagian akhir (tail) dari list. Prosesnya hampir mirip dengan fungsi `add_first()`, namun arah koneksi pointer berlawanan. Node baru dibuat dengan `prev` menunjuk ke node terakhir sebelumnya (`ptr_last`) dan `next` bernilai `NULL` karena node baru akan menjadi node terakhir.

- Fungsi `add_target()`

Berfungsi untuk menambahkan node baru di tengah list, tepat setelah node dengan nilai tertentu (target). Prosesnya dimulai dengan menelusuri list dari node pertama (`ptr_first`) hingga menemukan node yang memiliki nilai data sama dengan nilai target.

- Fungsi `view()`

Fungsi `view()` digunakan untuk menampilkan seluruh isi list dari node pertama hingga terakhir. Langkahnya dimulai dengan memeriksa apakah list kosong. Jika pointer

ptr_first bernilai NULL, maka program akan mencetak pesan “List kosong”.

- Fungsi delete_first()
Fungsi delete_first() berfungsi untuk menghapus node pertama pada list. Prosesnya diawali dengan memeriksa apakah list kosong. Jika ptr_first bernilai NULL, maka fungsi langsung berhenti karena tidak ada data yang bisa dihapus.
- Fungsi delete_last()
Fungsi delete_last() berfungsi untuk menghapus node terakhir dalam list. Sama seperti delete_first(), fungsi ini memeriksa apakah list kosong terlebih dahulu.
- Fungsi delete_target()
Fungsi delete_target() digunakan untuk menghapus node tertentu berdasarkan nilai data yang dicari. Langkah pertama adalah menelusuri node satu per satu mulai dari ptr_first hingga menemukan node yang memiliki data sama dengan targetValue.
- Fungsi edit_node()
Fungsi edit_node() digunakan untuk mengubah nilai data dari node tertentu. Program akan menelusuri setiap node dari awal hingga menemukan node dengan nilai data yang sama dengan targetValue.
- Fungsi main()
Bagian main() berfungsi untuk menjalankan semua operasi yang telah didefinisikan sebelumnya. Program akan menambahkan

beberapa node baru ke dalam list menggunakan fungsi `add_first()` dan `add_last()`. Setelah itu, list akan ditampilkan menggunakan fungsi `view()`. Kemudian, program melakukan penghapusan node pertama dengan `delete_first()`, lalu menampilkan hasilnya. Setelah itu dilakukan penghapusan node terakhir menggunakan `delete_last()`, kemudian ditampilkan lagi hasil list terkini. Selanjutnya, program menambahkan beberapa node baru dengan `add_last()` dan kembali menampilkan hasil list setelah penambahan. Terakhir, program menghapus node dengan nilai tertentu menggunakan fungsi `delete_target()`, lalu menampilkan hasil akhir dari list. Melalui urutan eksekusi ini, pengguna dapat melihat secara langsung bagaimana proses penambahan, penghapusan, dan tampilan data dalam struktur *doubly linked list* berjalan.

C. Unguided

1. Membuat playlist lagu dengan SLL

Source code doublyList.h

```
#ifndef DOUBLYLIST_H
#define DOUBLYLIST_H

#include <string>
using namespace std;

#define Nil nullptr

struct kendaraan {
    string nopol;
    string warna;
    int thnBuat;
};
```

```

});

typedef struct ElmList* address;

struct ElmList {
    kendaraan info;
    address next;
    address prev;
};

struct List {
    address first;
    address last;
};

void CreateList(List &L);
address alokasi(const kendaraan &x);
void dealokasi(address &P);
void insertLast(List &L, address P);
void printInfo(const List &L);
address findElm(const List &L, const string &nopol);
void deleteFirst(List &L, address &P);
void deletelast(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);

#endif

```

Source code doublyList.cpp

```

#include "Doublylist.h"
#include <iostream>
using namespace std;

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

address alokasi(const kendaraan &x) {
    address P = new ElmList;
    P->info = x;
    P->next = Nil;
    P->prev = Nil;
    return P;
}

```

```

void dealokasi(address &P) {
    if (P != Nil) {
        delete P;
        P = Nil;
    }
}

void insertLast(List &L, address P) {
    if (L.first == Nil) {
        L.first = P;
        L.last = P;
    } else {
        P->prev = L.last;
        L.last->next = P;
        L.last = P;
    }
}

void printInfo(const List &L) {
    if (L.first == Nil) {
        cout << "List kosong" << endl;
        return;
    }

    cout << "\nDATA LIST 1\n\n";
    address P = L.last;
    while (P != Nil) {
        cout << "no polisi : " << P->info.nopol << endl;
        cout << "warna      : " << P->info.warna << endl;
        cout << "tahun       : " << P->info.thnBuat << endl;
        P = P->prev;
        if (P != Nil) cout << endl;
    }
    cout << endl;
}

address findElm(const List &L, const string &nopol) {
    address P = L.first;
    while (P != Nil) {
        if (P->info.nopol == nobol) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

```

```

}

void deleteFirst(List &L, address &P) {
    if (L.first == Nil) {
        P = Nil;
    } else if (L.first == L.last) {
        P = L.first;
        L.first = Nil;
        L.last = Nil;
    } else {
        P = L.first;
        L.first = P->next;
        L.first->prev = Nil;
        P->next = Nil;
    }
}

void deleteLast(List &L, address &P) {
    if (L.first == Nil) {
        P = Nil;
    } else if (L.first == L.last) {
        P = L.first;
        L.first = Nil;
        L.last = Nil;
    } else {
        P = L.last;
        L.last = P->prev;
        L.last->next = Nil;
        P->prev = Nil;
    }
}

void deleteAfter(List &L, address Prec, address &P) {
    if (Prec == Nil || Prec->next == Nil) {
        P = Nil;
    } else {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != Nil) {
            P->next->prev = Prec;
        } else {
            L.last = Prec;
        }
        P->next = Nil;
        P->prev = Nil;
    }
}

```

Source code main.cpp

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

int main() {
    List L;
    CreateList(L);
    kendaraan k;
    address P;

    cout << "masukkan nomor polisi: D001\n";
    cout << "masukkan warna kendaraan: hitam\n";
    cout << "masukkan tahun kendaraan: 90\n\n";
    k.nopol = "D001"; k.warna = "hitam"; k.thnBuat = 90;
    if (findElm(L, k.nopol) == Nil)
        insertLast(L, alokasi(k));
    else
        cout << "nomor polisi sudah terdaftar\n\n";

    cout << "masukkan nomor polisi: D003\n";
    cout << "masukkan warna kendaraan: putih\n";
    cout << "masukkan tahun kendaraan: 70\n\n";
    k.nopol = "D003"; k.warna = "putih"; k.thnBuat = 70;
    if (findElm(L, k.nopol) == Nil)
        insertLast(L, alokasi(k));
    else
        cout << "nomor polisi sudah terdaftar\n\n";

    cout << "masukkan nomor polisi: D001\n";
    cout << "masukkan warna kendaraan: merah\n";
    cout << "masukkan tahun kendaraan: 80\n";
    if (findElm(L, "D001") == Nil)
        insertLast(L, alokasi(k));
    else
        cout << "nomor polisi sudah terdaftar\n\n";

    cout << "masukkan nomor polisi: D004\n";
    cout << "masukkan warna kendaraan: kuning\n";
    cout << "masukkan tahun kendaraan: 90\n\n";
    k.nopol = "D004"; k.warna = "kuning"; k.thnBuat = 90;
    if (findElm(L, k.nopol) == Nil)
        insertLast(L, alokasi(k));
    else
```

```

        cout << "nomor polisi sudah terdaftar\n\n";

    printInfo(L);

    cout << "Masukkan Nomor Polisi yang dicari : D001\n";
    address Q = findElm(L, "D001");
    if (Q != Nil) {
        cout << "\nNomor Polisi : " << Q->info.nopol << endl;
        cout << "Warna          : " << Q->info.warna << endl;
        cout << "Tahun          : " << Q->info.thnBuat << endl;
    } else {
        cout << "Data tidak ditemukan.\n";
    }

    cout << "\nMasukkan Nomor Polisi yang akan dihapus :
D003\n";
    Q = findElm(L, "D003");
    if (Q != Nil) {
        cout << "Data dengan nomor polisi D003 berhasil
dihapus.\n";
        address removed = Nil;
        if (Q == L.first)
            deleteFirst(L, removed);
        else if (Q == L.last)
            deleteLast(L, removed);
        else
            deleteAfter(L, Q->prev, removed);
        dealokasi(removed);
    } else {
        cout << "Data tidak ditemukan.\n";
    }
    printInfo(L);

    return 0;
}

```


Screenshot Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS I:\ATELYU\semester_3\struktur_data_ptk\modul6\unguided> g++ main.cpp doublyList.cpp
PS I:\ATELYU\semester_3\struktur_data_ptk\modul6\unguided> ./a
masukkan nomor polisi: D001
masukkan warna kendaraan: hitam
masukkan tahun kendaraan: 90

masukkan nomor polisi: D003
masukkan warna kendaraan: putih
masukkan tahun kendaraan: 70

masukkan nomor polisi: D001
masukkan warna kendaraan: merah
masukkan tahun kendaraan: 80
nomor polisi sudah terdaftar

masukkan nomor polisi: D004
masukkan warna kendaraan: kuning
masukkan tahun kendaraan: 90

DATA LIST 1

no polisi : D004
warna      : kuning
tahun      : 90

no polisi : D003
warna      : putih
tahun      : 70

no polisi : D001
warna      : hitam
tahun      : 90

Masukkan Nomor Polisi yang dicari : D001

Nomor Polisi : D001
Warna         : hitam
Tahun         : 90

Masukkan Nomor Polisi yang akan dihapus : D003
Data dengan nomor polisi D003 berhasil dihapus.

DATA LIST 1

no polisi : D004
warna      : kuning
tahun      : 90

no polisi : D001
warna      : hitam
tahun      : 90

PS I:\ATELYU\semester_3\struktur_data_ptk\modul6\unguided> 
```

Deskripsi:

Program diatas adalah impelementasi ADT dan Doubly Linked List, digunakan untku menyimpan kumpulan data kendaraan secara dinamis untuk menambah, mencari, menampilkan, dan menghapus data. Program berjalan dengan Langkah kerja sebagai berikut:

- Source code doublyList.h

File ini berisi deklarasi tipe data dan fungsi utama yang digunakan dalam doubly linked list. Terdapat beberapa struktur yaitu:

- Struct kendaraan berisi informasi kendaraan berupa nomor polisi (string nopol), warna kendaraan (string warna), dan tahun pembuatan (int thnBuat).
 - Struct ElmList berfungsi sebagai node dalam list, yang memiliki tiga bagian, yaitu info (menyimpan data kendaraan), next (penunjuk ke node berikutnya), dan prev (penunjuk ke node sebelumnya).
 - Struct List berfungsi menyimpan dua pointer penting, yaitu first untuk menunjuk ke node pertama dan last untuk menunjuk ke node terakhir.
- Source code doublyList.cpp

File ini berisi implementasi dari seluruh prosedur dan fungsi yang dideklarasikan di file header.

- Pertama createList() untuk menginisialisasi list agar kosong dengan mengatur pointer first dan last menjadi NULL.
- Alokasi() untuk membuat node baru pada memori dengan data tertentu dan dealokasi() untuk menghapus node dari memori agar data tidak bocor.

- InsertLast() untuk menambahkan node baru di bagian akhir list dan findElm() untuk mencari data berdasarkan nomor polisi.
 - Kemudian prosedur penghapusan ada deleteFirst() untuk menghapus node pertama dan memperbarui pointer first, deleteLast() untuk menghapus node terakhir dan memperbarui pointer last, dan deleteAfter() untuk menghapus node tengah list dengan memperbarui hubungan pointer next dan prev antar node.
 - printInfo() digunakan untuk menampilkan seluruh data kendaraan dari node terakhir sampai node pertama.
- Source code main.cpp

File main.cpp merupakan program utama untuk menjalankan seluruh operasi dan fungsi yang telah di definisikan dalam ADT, berikut adalah langkah kerjanya.

 - Pertama Program memanggil CreateList() untuk membuat list kosong. Kemudian beberapa data kendaraan ditambahkan menggunakan insertLast(). Pada saat yang sama, dilakukan pengecekan dengan findElm() agar nomor polisi yang sama tidak dimasukkan dua kali.
 - Kemudian setelah penambahan selesai, program panggil printInfo() untuk

menampilkan seluruh isi list sesuai dengan format.

- Pada bagian pencarian data kendaraan Program menampilkan proses pencarian menggunakan fungsi `findElm()`. Sebagai contoh, program mencari data dengan nomor polisi “D001”, kemudian menampilkan hasil ke terminal.
- Terakhir penghapusan data kendaraan yaitu program menghapus data kendaraan dengan nomor polisi tertentu, misalnya “D003”. Proses ini menggunakan kombinasi fungsi `findElm()` untuk menemukan data dan `deleteAfter()` atau `deleteFirst()` atau `deleteLast()` tergantung posisi node. Setelah data dihapus, `printInfo()` dipanggil kembali untuk menampilkan list terbaru.

D. Kesimpulan

Berdasarkan hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa Doubly Linked List merupakan salah satu struktur data dinamis yang sangat fleksibel dalam pengelolaan data. Struktur ini memungkinkan setiap elemen (node) terhubung dua arah melalui pointer prev dan next, sehingga proses penelusuran dapat dilakukan dari kedua sisi, baik dari depan maupun dari belakang.

Melalui implementasi yang dibuat pada praktikum ini, berbagai operasi dasar pada Doubly Linked List berhasil dilakukan, seperti menambahkan data di awal maupun di akhir list, menyisipkan data setelah node tertentu, mencari data berdasarkan kunci tertentu, serta menghapus node pada berbagai posisi baik di awal, akhir, maupun di tengah list. Semua operasi tersebut dapat dilakukan tanpa harus memindahkan elemen-elemen lain sebagaimana pada struktur array, sehingga menjadikan Doubly Linked List lebih efisien dalam situasi yang membutuhkan perubahan ukuran data secara dinamis. Selain itu, penggunaan pembagian program menjadi tiga berkas utama (Doublylist.h, Doublylist.cpp, dan main.cpp) menunjukkan penerapan konsep pemrograman modular yang baik, di mana deklarasi dan implementasi fungsi dipisahkan untuk memudahkan pengembangan, perawatan, serta pemahaman alur program.

Secara keseluruhan, praktikum ini memperkuat pemahaman mengenai pengelolaan pointer, memori dinamis, serta konsep manipulasi node dalam struktur data.

E. Referensi

- Banjarnahor, J. (2022). *Pemanfaatan Link List untuk Mengatasi Database Tidak Normal*. LoFian: Jurnal Teknologi Informasi, Universitas Muhammadiyah Bandung. Diakses dari <https://ejournal.umbp.ac.id/index.php/lofian/article/view/183>
- Mbejo, M. T. (2025). *Analisis Struktur Data Linked List dalam Pengolahan Data Dinamis*. RCF Indonesia Journal of Information Technology, 5(1). Diakses dari <https://rcf-indonesia.org/jurnal/index.php/jsit/article/view/591>
- Winarsih, S. M. S. (2022). *Implementasi dan Pengujian Struktur Data Berbasis Acuan*. TIKomSiN: Jurnal Teknologi Informasi, 3(2). Diakses dari <https://p3m.sinus.ac.id/jurnal/index.php/TIKomiN/article/viewFile/631/492>