

**LAPORAN PRAKTIKUM
STRUKTUR DATA DAN ALGORITMA**

**MODUL III
SINGLE AND DOUBLE LINKED LIST**



Disusun Oleh :
DIO GILBRAN PRAMANA
NIM : 2311102062

Dosen
WAHYU ANDI SAPUTRA, S.Pd., M.Eng.

**PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2024**

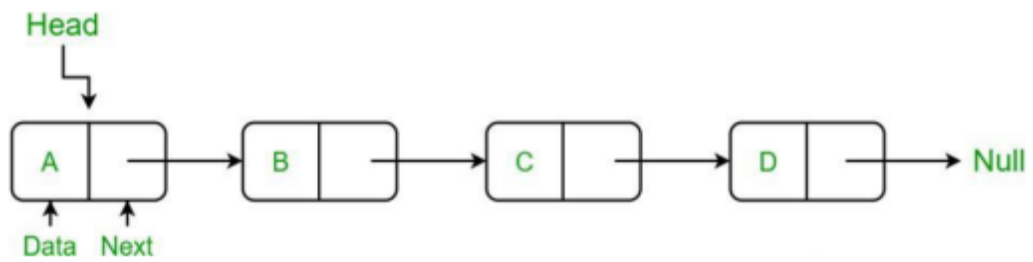
A. DASAR TEORI

Pengertian Linked List

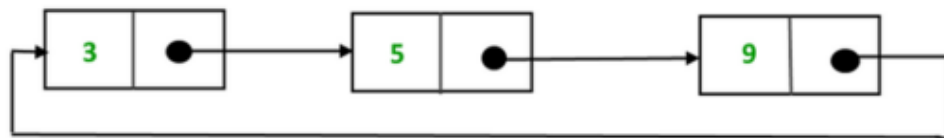
Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas adalah senarai berkait (linked list). Suatu senarai berkait (linked list) adalah suatu simpul (node) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau class. Simpul harus mempunyai satu atau lebih elemen struktur atau class yang berisi data. Secara teori, linked list adalah sejumlah node yang dihubungkan secara linier dengan bantuan pointer. Senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada runtime. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompilasi, hal ini bisa merupakan suatu atribut yang baik juga. Setiap node akan berbentuk struct dan memiliki satu buah field bertipe struct yang sama, yang berfungsi sebagai pointer. Dalam menghubungkan setiap node, kita dapat menggunakan cara first-create-first-access ataupun first-create-lastaccess. Yang berbeda dengan deklarasi struct sebelumnya adalah satu field bernama next, yang bertipe struct tnode. Hal ini sekilas dapat membingungkan. Namun, satu hal yang jelas, variabel next ini akan menghubungkan kita dengan node di sebelah kita, yang juga bertipe struct tnode. Hal inilah yang menyebabkan next harus bertipe struct tnode.

a) Single Linked List

Linked List merupakan suatu bentuk struktur data yang berisi kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung, dinamis, dan terbatas. Setiap elemen dalam linked list dihubungkan ke elemen lain melalui pointer. Masing-masing komponen sering disebut dengan simpul atau node atau verteks. Pointer adalah alamat elemen. Setiap simpul pada dasarnya dibagi atas dua bagian pertama disebut bagian isi atau informasi atau data yang berisi nilai yang disimpan oleh simpul. Bagian kedua disebut bagian pointer yang berisi alamat dari node berikutnya atau sebelumnya. Dengan menggunakan struktur seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya. Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head dan elemen terakhir dari suatu list disebut tail.



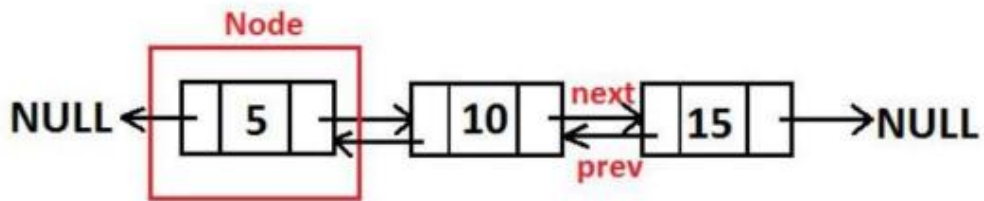
Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya lebih efisien dalam penggunaan memori dibandingkan dengan jenis Linked List lainnya, seperti Double Linked List dan Circular Linked List. Single linked list yang kedua adalah circular linked list. Perbedaan circular linked list dan non circular linked adalah penunjuk next pada node terakhir pada circular linked list akan selalu merujuk ke node pertama.



b) Double Linked List

Double Linked List adalah struktur data Linked List yang mirip dengan Single Linked List, namun dengan tambahan satu pointer tambahan pada setiap simpul yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul mana saja secara efisien. Setiap simpul pada Double Linked List memiliki tiga elemen penting, yaitu elemen data (biasanya berupa nilai), pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya. Keuntungan dari Double Linked List adalah memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul dimana saja dengan efisien, sehingga sangat berguna dalam implementasi beberapa algoritma yang membutuhkan operasi tersebut. Selain itu, Double Linked List juga memungkinkan kita untuk melakukan traversal pada list baik dari depan (head) maupun dari belakang (tail) dengan mudah. Namun, kekurangan dari Double Linked List adalah penggunaan memori yang lebih besar dibandingkan dengan Single Linked List, karena setiap simpul membutuhkan satu pointer tambahan. Selain itu, Double Linked List juga membutuhkan waktu eksekusi yang lebih lama dalam operasi penambahan dan penghapusan jika dibandingkan dengan Single Linked List.

Representasi sebuah double linked list dapat dilihat pada gambar berikut ini:



Di dalam sebuah linked list, ada 2 pointer yang menjadi penunjuk utama, yakni pointer HEAD yang menunjuk pada node pertama di dalam linked list itu sendiri dan pointer TAIL yang menunjuk pada node paling akhir di dalam linked list. Sebuah linked list dikatakan kosong apabila isi pointer head adalah NULL. Selain itu, nilai pointer prev dari HEAD selalu NULL, karena merupakan data pertama. Begitu pula dengan pointer next dari TAIL yang selalu bernilai NULL sebagai penanda data terakhir.

B. GUIDED

GUIDED 1

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

// Deklarasi Struct Node
struct Node {
    int data;
    Node* next;
};

Node* head;
Node* tail;

// Inisialisasi Node
void init() {
    head = NULL;
    tail = NULL;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == NULL;
}

// Tambah Node di depan
void insertDepan(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
```

```

        if (isEmpty()) {
            head = tail = baru;
        } else {
            baru->next = head;
            head = baru;
        }
    }

// Tambah Node di belakang
void insertBelakang(int nilai) {
    Node* baru = new Node;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung jumlah Node di list
int hitungList() {
    Node* hitung = head;
    int jumlah = 0;
    while (hitung != NULL) {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}

// Tambah Node di posisi tengah
void insertTengah(int data, int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi diluar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node* baru = new Node();
        baru->data = data;
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1) {
            bantu = bantu->next;
            nomor++;
        }
    }
}

```

```

        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Node di depan
void hapusDepan() {
    if (!isEmpty()) {
        Node* hapus = head;
        if (head->next != NULL) {
            head = head->next;
            delete hapus;
        } else {
            head = tail = NULL;
            delete hapus;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Node di belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head != tail) {
            Node* hapus = tail;
            Node* bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            tail = bantu;
            tail->next = NULL;
            delete hapus;
        } else {
            head = tail = NULL;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Node di posisi tengah
void hapusTengah(int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi diluar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    }
}

```

```

    } else {
        Node* hapus;
        Node* bantu = head;
        for (int nomor = 1; nomor < posisi - 1; nomor++) {
            bantu = bantu->next;
        }
        hapus = bantu->next;
        bantu->next = hapus->next;
        delete hapus;
    }
}

// Ubah data Node di depan
void ubahDepan(int data) {
    if (!isEmpty()) {
        head->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di posisi tengah
void ubahTengah(int data, int posisi) {
    if (!isEmpty()) {
        if (posisi < 1 || posisi > hitungList()) {
            cout << "Posisi di luar jangkauan" << endl;
        } else if (posisi == 1) {
            cout << "Posisi bukan posisi tengah" << endl;
        } else {
            Node* bantu = head;
            for (int nomor = 1; nomor < posisi; nomor++) {
                bantu = bantu->next;
            }
            bantu->data = data;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di belakang
void ubahBelakang(int data) {
    if (!isEmpty()) {
        tail->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

```

```

}

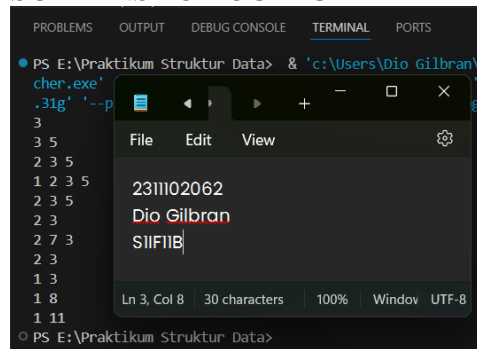
// Hapus semua Node di list
void clearList() {
    Node* bantu = head;
    while (bantu != NULL) {
        Node* hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}

// Tampilkan semua data Node di list
void tampil() {
    if (!isEmpty()) {
        Node* bantu = head;
        while (bantu != NULL) {
            cout << bantu->data << " ";
            bantu = bantu->next;
        }
        cout << endl;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

int main() {
    init();
    insertDepan(3); tampil();
    insertBelakang(5); tampil();
    insertDepan(2); tampil();
    insertDepan(1); tampil();
    hapusDepan(); tampil();
    hapusBelakang(); tampil();
    insertTengah(7, 2); tampil();
    hapusTengah(2); tampil();
    ubahDepan(1); tampil();
    ubahBelakang(8); tampil();
    ubahTengah(11, 2); tampil();
    return 0;
}

```


SCREENSHOT OUTPUT



```
PS E:\Praktikum Struktur Data> & 'c:\Users\Dio Gilbran\cher.exe' .31g' '--p
3
3 5
2 3 5
1 2 3 5
2 3 5
2 3
2 7 3
2 3
1 3
1 8
1 11
PS E:\Praktikum Struktur Data>
```

DESKRIPSI

Program ini mengimplementasikan operasi dasar pada linked list menggunakan C++. Linked list diwakili oleh struktur Node yang menyimpan data dan pointer ke Node berikutnya. Program menggunakan pointer global head dan tail untuk melacak Node pertama dan terakhir. Fungsi-fungsi utama meliputi: inisialisasi (init), penambahan Node di depan (insertDepan), belakang (insertBelakang), dan posisi tertentu (insertTengah), penghapusan Node di depan (hapusDepan), belakang (hapusBelakang), dan posisi tertentu (hapusTengah), penghitungan jumlah Node (hitungList), pengubahan nilai Node di depan, tengah, dan belakang (ubahDepan, ubahTengah, ubahBelakang), pembersihan seluruh linked list (clearList), dan penampilan semua data dalam linked list (tampil). Fungsi main menguji semua operasi ini dengan contoh-contoh sederhana. Program ini memberikan gambaran lengkap tentang operasi dasar pada linked list dalam C++.

GUIDED 2

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

class Node
{
public:
    int data;
    Node *prev;
    Node *next;
};

class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;

    DoublyLinkedList()
```

```

{
    head = nullptr;
    tail = nullptr;
}

void push(int data)
{
    Node *newNode = new Node;
    newNode->data = data;
    newNode->prev = nullptr;
    newNode->next = head;

    if (head != nullptr)
    {
        head->prev = newNode;
    }
    else
    {
        tail = newNode;
    }

    head = newNode;
}

void pop()
{
    if (head == nullptr)
    {
        return;
    }
    Node *temp = head;
    head = head->next;

    if (head != nullptr)
    {
        head->prev = nullptr;
    }
    else
    {
        tail = nullptr;
    }

    delete temp;
}

bool update(int oldData, int newData)
{

```

```

        Node *current = head;

        while (current != nullptr)
        {
            if (current->data == oldData)
            {
                current->data = newData;
                return true;
            }
            current = current->next;
        }
        return false;
    }

    void deleteAll()
    {
        Node *current = head;
        while (current != nullptr)
        {
            Node *temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    void display()
    {
        Node *current = head;
        while (current != nullptr)
        {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main()
{
    DoublyLinkedList list;
    while (true)
    {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
    }
}

```

```
cout << "4. Clear data" << endl;
cout << "5. Display data" << endl;
cout << "6. Exit" << endl;

int choice;
cout << "Enter your choice: ";
cin >> choice;

switch (choice)
{
case 1:
{
    int data;
    cout << "Enter data to add: ";
    cin >> data;
    list.push(data);
    break;
}
case 2:
{
    list.pop();
    break;
}
case 3:
{
    int oldData, newData;
    cout << "Enter old data: ";
    cin >> oldData;
    cout << "Enter new data: ";
    cin >> newData;
    bool updated = list.update(oldData, newData);
    if (!updated)
    {
        cout << "Data not found" << endl;
    }
    break;
}
case 4:
{
    list.deleteAll();
    break;
}
case 5:
{
    list.display();
    break;
}
}
```

```

        case 6:
        {
            return 0;
        }
        default:
        {
            cout << "Invalid choice" << endl;
            break;
        }
    }
    return 0;
}

```

SCREENSHOT OUTPUT

1.

Terminal output for screenshot 1:

```

PS E:\Praktikum Struktur Data> & 'c:\User\her.exe' '--stdin-Microsoft-MIEngine-In-2.fo' '--pid-Microsoft-MIEngine-Pid-ze2pcx
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 7
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
7 4

```

Visual editor overlay shows:

```

2311102062
Dio Gilbran
SIIF11B
Ln 3, Col 8 | 30 cha

```

2.

Terminal output for screenshot 2:

```

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 7
Enter new data: 66
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
66 4

```

Visual editor overlay shows:

```

2311102062
Dio Gilbran
SIIF11B
Ln 3, Col 8 | 30 cha

```

3.

Terminal output for screenshot 3:

```

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
1. Add data

```

Visual editor overlay shows:

```

2311102062
Dio Gilbran
SIIF11B
Ln 3, Col 8 | 30 cha

```

DESKRIPSI

Program ini mengimplementasikan operasi dasar pada doubly linked list menggunakan C++. Kelas Node menyimpan data dan pointer ke Node sebelumnya dan berikutnya, sedangkan kelas DoublyLinkedList mengelola operasi pada linked list, termasuk penambahan Node di depan (push), penghapusan Node di depan (pop), pembaruan nilai Node (update), penghapusan semua Node (deleteAll), dan penampilan semua data dalam linked list (display). Fungsi main menyediakan antarmuka pengguna untuk mengakses operasi-operasi ini melalui menu interaktif, memungkinkan pengguna untuk menambah, menghapus, memperbarui, mengosongkan, dan menampilkan data dalam linked list. Program ini mendemonstrasikan penggunaan linked list ganda dengan berbagai operasi dasar yang diakses melalui input pengguna.

C. UNGUIDED

UNGUIDED 1

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

struct Node
{
    string nama;
    int usia;
    Node *next;
};
Node *head;
Node *tail;

void init()
{
    head = NULL;
    tail = NULL;
}

bool isEmpty()
{
    return head == NULL;
}

void insertDepan(string nama, int usia)
{
    Node *baru = new Node;
    baru->nama = nama;
    baru->usia = usia;
    baru->next = NULL;
    if (isEmpty())
    {
        head = tail = baru;
    }
}
```

```

        else
        {
            baru->next = head;
            head = baru;
        }
    }

void insertBelakang(string nama, int usia)
{
    Node *baru = new Node;
    baru->nama = nama;
    baru->usia = usia;
    baru->next = NULL;
    if (isEmpty())
    {
        head = tail = baru;
    }
    else
    {
        tail->next = baru;
        tail = baru;
    }
}

int hitungList()
{
    Node *hitung = head;
    int jumlah = 0;
    while (hitung != NULL)
    {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}

void insertTengah(string nama, int usia, int posisi)
{
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi diluar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else

```

```

    {
        Node *baru = new Node();
        baru->nama = nama;
        baru->usia = usia;
        Node *bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

void hapusTengah(string nama)
{
    if (!isEmpty())
    {
        Node *hapus;
        Node *bantu = head;
        Node *prev = nullptr;
        while (bantu != nullptr && bantu->nama != nama)
        {
            prev = bantu;
            bantu = bantu->next;
        }
        if (bantu == nullptr)
        {
            cout << "Nama tidak ditemukan dalam list" << endl;
            return;
        }
        if (prev != nullptr)
        {
            prev->next = bantu->next;
        }
        else
        {
            head = bantu->next;
        }
        if (bantu == tail)
        {
            tail = prev;
        }
        delete bantu;
    }
}

```



```

        cout << "Node dengan nama '" << nama << "' berhasil
dihapus" << endl;
    }
    else
    {
        cout << "List masih kosong" << endl;
    }
}

void ubahTengah(string nama, int usia, string nama_lama)
{
    if (!isEmpty())
    {
        Node *bantu = head;
        while (bantu != nullptr && bantu->nama != nama_lama)
        {
            bantu = bantu->next;
        }
        if (bantu != nullptr)
        {
            bantu->nama = nama;
            bantu->usia = usia;
            cout << "Node dengan nama '" << nama_lama << "'
berhasil diubah menjadi '" << nama << "' dengan usia baru " <<
usia << endl;
        }
        else
        {
            cout << "Node dengan nama '" << nama_lama << "' tidak
ditemukan" << endl;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

void tampil()
{
    if (!isEmpty())
    {
        Node *bantu = head;
        while (bantu != NULL)
        {
            cout << bantu->nama << " " << bantu->usia << endl;
            bantu = bantu->next;
        }
    }
}

```

```

    }
    cout << endl;
}
else
{
    cout << "List masih kosong!" << endl;
}
}

int main()
{
    init();
    string nama;
    int usia, choice;
    cout << "Masukkan nama anda: ";
    cin >> nama;
    cout << "Masukkan usia anda: ";
    cin >> usia;
    insertBelakang(nama, usia);
    while (true)
    {
        cout << "1. Tambah Data di depan" << endl;
        cout << "2. Tambah Data di belakang" << endl;
        cout << "3. Tambah Data di posisi tengah" << endl;
        cout << "4. Hapus Data berdasarkan nama" << endl;
        cout << "5. Ubah data Data berdasarkan nama" << endl;
        cout << "6. Tampilkan semua Data di list" << endl;
        cout << "7. Keluar" << endl;
        cout << "Masukkan pilihan: ";
        cin >> choice;
        switch (choice)
        {
            case 1:
            {
                cout << "Masukkan nama: ";
                cin >> nama;
                cout << "Masukkan usia: ";
                cin >> usia;
                insertDepan(nama, usia);
                break;
            }
            case 2:
            {
                cout << "Masukkan nama: ";
                cin >> nama;
                cout << "Masukkan usia: ";
                cin >> usia;
            }
        }
    }
}

```

```

        insertBelakang(nama, usia);
        break;
    }
    case 3:
    {
        int posisi;
        cout << "Masukkan nama: ";
        cin >> nama;
        cout << "Masukkan usia: ";
        cin >> usia;
        cout << "Masukkan posisi: ";
        cin >> posisi;
        insertTengah(nama, usia, posisi);
        break;
    }
    case 4:
    {
        cout << "Masukkan nama yang ingin dihapus: ";
        cin >> nama;
        hapusTengah(nama);
        break;
    }
    case 5:
    {
        string nama_lama;
        cout << "Masukkan nama lama: ";
        cin >> nama_lama;
        cout << "Masukkan nama baru: ";
        cin >> nama;
        cout << "Masukkan usia baru: ";
        cin >> usia;
        ubahTengah(nama, usia, nama_lama);
        break;
    }
    case 6:
    {
        tampil();
        break;
    }
    case 7:
    {
        return 0;
    }
    default:
    {
        cout << "Pilihan tidak valid" << endl;
        break;
    }
}

```

```

    }
    }
}
return 0;
}

```

SCREENSHOT OUTPUT

1.

```

Masukkan usia: 18
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 2
Masukkan nama: Karin
Masukkan usia: 18
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 6
Dio 19
Jane 20
Michael 18
Yusuke 19
Akechi 20
Hoshino 18
Karin 18

```

2.

```

1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 4
Masukkan nama yang ingin dihapus: Akechi
Node dengan nama 'Akechi' berhasil dihapus
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 6
Dio 19
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18

```

3.

```

1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 3
Masukkan nama: Futaba
Masukkan usia: 18
Masukkan posisi: 2
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 6
Dio 19
Futaba 18
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18

```

4.

```

1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 1
Masukkan nama: Igor
Masukkan usia: 20
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 6
Igor 20
Dio 19
Futaba 18
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18

```

5.

```

1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 5
Masukkan nama lama: Michael
Masukkan nama baru: Reyn
Masukkan usia baru: 18
Node dengan nama 'Michael' berhasil diubah menjadi 'Reyn' dengan usia baru 18
1. Tambah Data di depan
2. Tambah Data di belakang
3. Tambah Data di posisi tengah
4. Hapus Data berdasarkan nama
5. Ubah data Data berdasarkan nama
6. Tampilkan semua Data di list
7. Keluar
Masukkan pilihan: 6
Igor 20
Dio 19
Futaba 18
Jane 20
Reyn 18
Yusuke 19
Hoshino 18
Karin 18

```

DESKRIPSI

Program ini adalah implementasi dari struktur data linked list sederhana dalam C++ yang menyediakan beberapa fungsi untuk memanipulasi data di dalamnya. Program ini dapat menambahkan node baru di depan, di belakang, atau di posisi tertentu dalam list, menghapus node berdasarkan nama, mengubah data dalam node yang sudah ada, serta menampilkan semua data dalam list. Struktur dasar dari linked list diwakili oleh struct Node, yang memiliki atribut nama, usia, dan pointer next ke node berikutnya. Program ini juga menyediakan antarmuka pengguna berbasis teks yang memungkinkan pengguna untuk memilih operasi yang ingin dilakukan, seperti menambah, menghapus, mengubah, atau menampilkan data dalam list.

UNGUIDED 2

```
#include <iostream>
#include <string>
using namespace std;
class Node
{
public:
    string namaProduk;
    double harga;
    Node *prev;
    Node *next;
};
class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void push(string namaProduk, double harga)
    {
        Node *newNode = new Node;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        newNode->prev = nullptr;
        newNode->next = nullptr;
        if (head == nullptr)
        {
            head = newNode;
            tail = newNode;
        }
    }
}
```

```

        else
        {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
}

void pushAt(int pos, string namaProduk, double harga)
{
    if (pos < 1)
    {
        cout << "Posisi tidak valid" << endl;
        return;
    }
    Node *newNode = new Node;
    newNode->namaProduk = namaProduk;
    newNode->harga = harga;
    if (pos == 1)
    {
        newNode->next = head;
        newNode->prev = nullptr;
        if (head != nullptr)
        {
            head->prev = newNode;
        }
        else
        {
            tail = newNode;
        }
        head = newNode;
        return;
    }
    Node *current = head;
    int count = 1;
    while (current != nullptr && count < pos - 1)
    {
        current = current->next;
        count++;
    }
    if (current == nullptr)
    {
        cout << "Posisi tidak valid" << endl;
        return;
    }
    newNode->next = current->next;
    newNode->prev = current;
    if (current->next != nullptr)

```

```

        {
            current->next->prev = newNode;
        }
        else
        {
            tail = newNode;
        }
        current->next = newNode;
    }
void pop()
{
    if (head == nullptr)
    {
        return;
    }
    Node *temp = head;
    head = head->next;
    if (head != nullptr)
    {
        head->prev = nullptr;
    }
    else
    {
        tail = nullptr;
    }
    delete temp;
}
void popAt(int pos)
{
    if (pos < 1)
    {
        cout << "Posisi tidak valid" << endl;
        return;
    }
    if (pos == 1)
    {
        if (head == nullptr)
        {
            return;
        }
        Node *temp = head;
        head = head->next;
        if (head != nullptr)
        {
            head->prev = nullptr;
        }
        else

```

```

        {
            tail = nullptr;
        }
        delete temp;
        return;
    }
    Node *current = head;
    int count = 1;
    while (current != nullptr && count < pos)
    {
        current = current->next;
        count++;
    }
    if (current == nullptr)
    {
        cout << "Posisi tidak valid" << endl;
        return;
    }
    if (current->next != nullptr)
    {
        current->next->prev = current->prev;
    }
    else
    {
        tail = current->prev;
    }
    current->prev->next = current->next;
    delete current;
}

bool update(string oldNamaProduk, string newNamaProduk, double
newHarga)
{
    Node *current = head;
    while (current != nullptr)
    {
        if (current->namaProduk == oldNamaProduk)
        {
            current->namaProduk = newNamaProduk;
            current->harga = newHarga;
            return true;
        }
        current = current->next;
    }
    return false;
}

void deleteAll()
{

```



```

        Node *current = head;
        while (current != nullptr)
        {
            Node *temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }
    void display()
    {
        cout << "Nama Produk Harga" << endl;
        Node *current = head;
        while (current != nullptr)
        {
            cout << current->namaProduk << " " << current->harga
                << endl;
            current = current->next;
        }
        cout << endl;
    }
};

int main()
{
    DoublyLinkedList list;
    while (true)
    {
        cout << "Toko Skincare Purwokerto" << endl;
        cout << "1. Tambah Data Produk" << endl;
        cout << "2. Hapus Data Produk" << endl;
        cout << "3. Update Data Produk" << endl;
        cout << "4. Tambah Data Produk Urutan Tertentu" << endl;
        cout << "5. Hapus Data Produk Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data Produk" << endl;
        cout << "7. Tampilkan Data" << endl;
        cout << "8. Exit" << endl;
        int choice;
        cout << "Masukkan pilihan Anda: ";
        cin >> choice;
        switch (choice)
        {
            case 1:
            {
                string namaProduk;
                double harga;
                cout << "Masukkan Nama Produk: ";
            }
        }
    }
}

```

```

        cin.ignore();
        getline(cin, namaProduk);
        cout << "Masukkan Harga Produk: ";
        cin >> harga;
        list.push(namaProduk, harga);
        break;
    }
    case 2:
    {
        list.pop();
        break;
    }
    case 3:
    {
        string oldNamaProduk, newNamaProduk;
        double newHarga;
        cout << "Masukkan Nama Produk Lama: ";
        cin.ignore();
        getline(cin, oldNamaProduk);
        cout << "Masukkan Nama Produk Baru: ";
        getline(cin, newNamaProduk);
        cout << "Masukkan Harga Produk Baru: ";
        cin >> newHarga;
        bool updated = list.update(oldNamaProduk,
newNamaProduk, newHarga);
        if (!updated)
        {
            cout << "Produk tidak ditemukan" << endl;
        }
        break;
    }
    case 4:
    {
        int pos;
        string namaProduk;
        double harga;
        cout << "Masukkan posisi: ";
        cin >> pos;
        cout << "Masukkan Nama Produk: ";
        cin.ignore();
        getline(cin, namaProduk);
        cout << "Masukkan Harga Produk: ";
        cin >> harga;
        list.pushAt(pos, namaProduk, harga);
        break;
    }
    case 5:

```

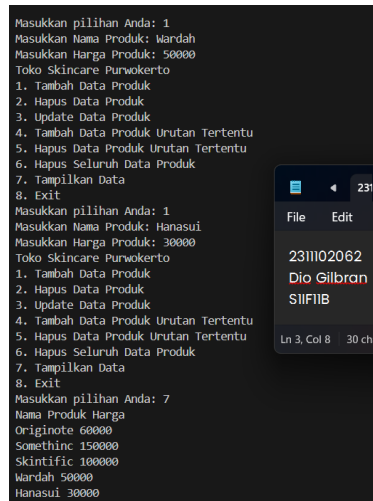
```

    {
        int pos;
        cout << "Masukkan posisi untuk menghapus: ";
        cin >> pos;
        list.popAt(pos);
        break;
    }
    case 6:
    {
        list.deleteAll();
        break;
    }
    case 7:
    {
        list.display();
        break;
    }
    case 8:
    {
        return 0;
    }
    default:
    {
        cout << "Pilihan tidak valid" << endl;
        break;
    }
}
return 0;
}

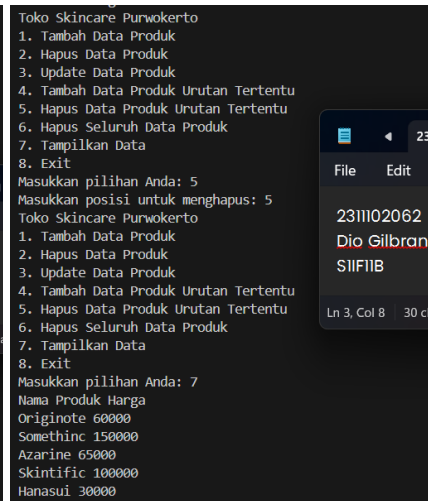
```

SCREENSHOT OUTPUT

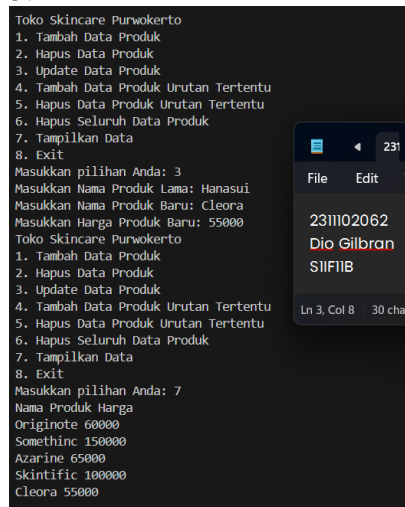
1.



2.



3.



```
Toko Skincare Purwokerto
1. Tambah Data Produk
2. Hapus Data Produk
3. Update Data Produk
4. Tambah Data Produk Urutan Tertentu
5. Hapus Data Produk Urutan Tertentu
6. Hapus Seluruh Data Produk
7. Tampilkan Data
8. Exit
Masukkan pilihan Anda: 3
Masukkan Nama Produk Lama: Hanasui
Masukkan Nama Produk Baru: Cleora
Masukkan Harga Produk Baru: 55000
Toko Skincare Purwokerto
1. Tambah Data Produk
2. Hapus Data Produk
3. Update Data Produk
4. Tambah Data Produk Urutan Tertentu
5. Hapus Data Produk Urutan Tertentu
6. Hapus Seluruh Data Produk
7. Tampilkan Data
8. Exit
Masukkan pilihan Anda: 7
Nama Produk Harga
Originote 60000
Somethinc 150000
Azarine 65000
Skintific 100000
Cleora 55000
```

DESKRIPSI

Program ini merupakan implementasi dari struktur data daftar tertaut ganda untuk mengelola data produk perawatan kulit di dalam toko. Setiap produk diwakili oleh node yang menyimpan nama dan harga produk, serta referensi ke node sebelumnya dan berikutnya. Program ini menawarkan berbagai fungsi seperti menambah (maju, mundur atau posisi tertentu), menghapus (maju, posisi tertentu atau semua data), memperbarui data produk, dan menampilkan semua produk dalam daftar. Antarmuka berbasis teks memungkinkan pengguna memilih tindakan yang ingin mereka lakukan melalui menu interaktif, memfasilitasi pengelolaan dinamis data produk di toko perawatan kulit.

D. KESIMPULAN

Linked List adalah struktur data dinamis yang terdiri dari node-node yang dihubungkan secara sekuensial melalui pointer. Terdapat dua jenis utama linked list: Single Linked List, yang hanya memiliki pointer ke node berikutnya, dan Double Linked List, yang memiliki pointer ke node sebelumnya dan berikutnya. Single Linked List lebih efisien dalam penggunaan memori karena hanya memerlukan satu pointer per node, namun Double Linked List lebih fleksibel karena memungkinkan traversal dan operasi penambahan serta penghapusan node dari kedua arah dengan lebih efisien. Meskipun Double Linked List menggunakan lebih banyak memori, keuntungan dalam efisiensi manipulasi data membuatnya sangat berguna dalam berbagai aplikasi algoritma.

E. REFERENSI

AsistenPraktikum, "ModulSingle and Double LinkedList", Learning ManagementSystem, 2024.

[ASD-Modul-6-Linked-List.pdf \(um.ac.id\)](https://um.ac.id/ASD-Modul-6-Linked-List.pdf) MODUL PRAKTIKUM ALGORITMA & STRUKTUR DATA Jurusan Teknik Elektro – Universitas Negeri Malang – 2016