

LAPORAN PRAKTIKUM

MODUL III LINKED LIST



Disusun oleh:
Muhammad Rifki Fadhilah
NIM: 2311102032

Dosen Pengampu:
Wahyu Andi Saputra, S.Pd., M.Eng.

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

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman

BAB II

DASAR TEORI

1. 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. Untuk menghubungkan satu node dengan node yang lainnya Linked List menggunakan pointer sebagai penunjuk node selanjutnya. Node sendiri merupakan sebuah struct yang terdiri dari beberapa field, minimal ada 2 buah field yaitu field untuk isi dari struct datanya sendiri, dan 1 field arbitrary bertipe pointer sebagai penunjuk node selanjutnya. 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. Untuk menghubungkan satu node dengan node yang lainnya Linked List menggunakan pointer sebagai penunjuk node selanjutnya. Node sendiri merupakan sebuah struct yang terdiri dari beberapa field, minimal ada 2 buah field yaitu field untuk isi dari struct datanya sendiri, dan 1 field arbitrary bertipe pointer sebagai penunjuk node selanjutnya. 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.

2. Double Linked List

Dalam pembahasan artikel sebelumnya telah diperkenalkan Single Linked List, yakni linked list dengan sebuah pointer penghubung. Dalam artikel ini, dibahas pula varian linked list dengan 2 pointer penunjuk, yakni Doubly linked list yang memiliki pointer penunjuk 2 arah, yakni ke arah node sebelum (previous/prev) dan node sesudah (next). Representasi sebuah doubly 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.

BAB III

GUIDED

1. Guided 1

Source code

```
#include <iostream>
using namespace std;
/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    // komponen/member
    int data;
    string kata;
    Node *next;
};
Node *head;
Node *tail;
// Inisialisasi Node
void init()
{
    head = NULL;
    tail = NULL;
}
// Pengecekan
bool isEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}
// Tambah Depan
void insertDepan(int nilai, string kata)
```

```

{
    // Buat Node baru
    Node *baru = new Node;
    baru->kata = kata;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
    else
    {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(int nilai, string kata)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->kata = kata;
    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
    else
    {
        tail->next = baru;
        tail = baru;
    }
}

```

```

    }
}
// Hitung Jumlah List
int hitungList()
{
    Node *hitung;
    hitung = head;
    int jumlah = 0;
    while (hitung != NULL)
    {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}
// Tambah Tengah
void insertTengah(int data, string kata, 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, *bantu;
        baru = new Node();

        baru->data = data;
        baru->kata = kata;
        // tranversing
    }
}

```

```

        bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

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

// Hapus Belakang

```



```

void hapusBelakang()
{
    Node *hapus;
    Node *bantu;
    if (isEmpty() == false)
    {
        if (head != tail)
        {
            hapus = tail;
            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 Tengah
void hapusTengah(int posisi)
{
    Node *hapus, *bantu, *bantu2;
    if (posisi < 1 || posisi > hitungList())
    {

```

```

        cout << "Posisi di luar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi)
        {
            if (nomor == posisi - 1)
            {
                bantu2 = bantu;
            }
            if (nomor == posisi)
            {
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        bantu2->next = bantu;
        delete hapus;
    }
}

// Ubah Depan
void ubahDepan(int data, string kata)
{
    if (isEmpty() == false)
    {
        head->data = data;
        head->kata = kata;
    }
}

```

```

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

// Ubah Tengah
void ubahTengah(int data, string kata, int posisi)
{
    Node *bantu;
    if (isEmpty() == false)
    {
        if (posisi < 1 || posisi > hitungList())
        {
            cout << "Posisi di luar jangkauan" << endl;
        }
        else if (posisi == 1)
        {
            cout << "Posisi bukan posisi tengah" << endl;
        }
        else
        {
            bantu = head;
            int nomor = 1;
            while (nomor < posisi)
            {
                bantu = bantu->next;
                nomor++;
            }
            bantu->data = data;
            bantu->kata = kata;
        }
    }
    else

```

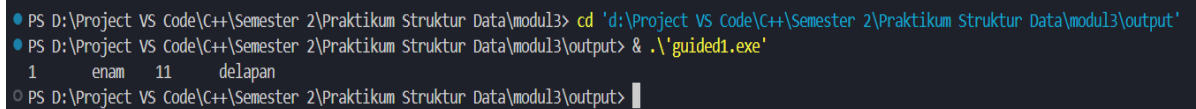
```
        {
            cout << "List masih kosong!" << endl;
        }
    }
    // Ubah Belakang
    void ubahBelakang(int data, string kata)
    {
        if (isEmpty() == false)
        {
            tail->data = data;
            tail->kata = kata;
        }
        else
        {
            cout << "List masih kosong!" << endl;
        }
    }
    // Hapus List
    void clearList()
    {
        Node *bantu, *hapus;
        bantu = head;
        while (bantu != NULL)
        {
            hapus = bantu;
            bantu = bantu->next;
            delete hapus;
        }
        head = tail = NULL;
        cout << "List berhasil terhapus!" << endl;
    }
    // Tampilkan List
    void tampil()
    {
```

```

Node *bantu;
bantu = head;
if (isEmpty() == false)
{
    while (bantu != NULL)
    {
        cout << bantu->data << "\t";
        cout << bantu->kata << "\t";
        bantu = bantu->next;
    }
    cout << endl;
}
else
{
    cout << "List masih kosong!" << endl;
}
}
int main()
{
    init();
    insertDepan(3, "satu");
    insertBelakang(5, "dua");
    insertDepan(2, "tiga");
    insertDepan(1, "empat");
    hapusDepan();
    hapusBelakang();
    insertTengah(7, "lima", 2);
    hapusTengah(2);
    ubahDepan(1, "enam");
    ubahBelakang(8, "tujuh");
    ubahTengah(11, "delapan", 2);
    tampil();
    return 0;
}

```

Screenshoot program



```
PS D:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul3> cd 'd:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul3\output'
PS D:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul3\output> & .\'guided1.exe'
1      enam      11      delapan
PS D:\Project VS Code\C++\Semester 2\Praktikum Struktur Data\modul3\output> █
```

Deskripsi program

Program diatas adalah implementasi sebuah linked list tunggal non circular. Program dimulai dengan mendefinisikan struct Node yang memiliki 3 komponen yaitu data(integer), kata(string), dan next(pointer Node ke elemen selanjutnya). Selanjutnya, program mendeklarasikan variabel global head dan tail yang merupakan pointer ke Node pertama dan terakhir. Fungsi init digunakan untuk menginisialisasi linked list dengan mengatur head dan tail menjadi NULL. Fungsi isEmpty digunakan untuk memeriksa apakah linked list kosong atau tidak. Fungsi insertDepan digunakan untuk menambahkan elemen baru di depan linked list. Jika linked list masih kosong, maka head dan tail akan menunjuk ke elemen baru tersebut. Jika tidak kosong, maka elemen baru akan ditambahkan di depan dan head akan diupdate. Fungsi insertBelakang mirip dengan insertDepan, namun elemen baru ditambahkan di belakang linked list. Fungsi hitungList digunakan untuk menghitung jumlah elemen dalam linked list. Fungsi insertTengah digunakan untuk menambahkan elemen baru di posisi tertentu dalam linked list. Fungsi ini memerlukan parameter posisi yang menentukan posisi dimana elemen baru akan ditambahkan. Fungsi hapusDepan digunakan untuk menghapus elemen pertama dari linked list. Fungsi hapusBelakang digunakan untuk menghapus elemen terakhir dari linked list. Fungsi hapusTengah digunakan untuk menghapus elemen pada posisi tertentu dalam linked list. Fungsi ubahDepan, ubahBelakang, dan ubahTengah digunakan untuk mengubah nilai data pada elemen pertama, terakhir, dan posisi tertentu dalam linked list. Fungsi

clearList digunakan untuk menghapus semua elemen dalam linked list. Fungsi tampil digunakan untuk menampilkan seluruh elemen dalam linked list. Di dalam fungsi main, program melakukan inisialisasi linked list menggunakan fungsi init, kemudian melakukan serangkaian operasi seperti menambah, menghapus, dan mengubah elemen dalam linked list, serta menampilkan isi linked list.

2. Guided 2

Source code

```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    string kata;
    Node *prev;
    Node *next;
};
class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void push(int data, string kata)
    {
```

```
Node *newNode = new Node;
newNode->data = data;
newNode->kata = kata;
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, string oldKata, string
newKata)
    {
        Node *current = head;
        while (current != nullptr)
        {
            if (current->data == oldData && current->kata ==
oldKata)
            {
                current->data = newData;
                current->kata = newKata;
                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 << " ";
        cout << current->kata << " ";
        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;
                string kata;
                cout << "Enter data to add: ";
                cin >> data;
                cout << "Enter kata to add: ";
                cin >> kata;
                list.push(data, kata);
                break;
            }

```

```
case 2:
{
    list.pop();
    break;
}
case 3:
{
    int oldData, newData;
    string oldKata, newKata;
    cout << "Enter old data: ";
    cin >> oldData;
    cout << "Enter new data: ";
    cin >> newData;
    bool updated = list.update(oldData,newData, oldKata,
newKata);
    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 program

```
PS D:\Project VS Code\C++\Se
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: 17
Enter kata to add: Agustus
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: 18
Enter kata to add: Maret
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
18 Maret 17 Agustus
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 
```

Deskripsi program

Program ini adalah implementasi dari sebuah Doubly Linked List, yang merupakan struktur data linear mirip dengan Linked List, tetapi setiap node memiliki dua pointer, yaitu pointer ke node sebelumnya (prev) dan node

selanjutnya (next). Pada awalnya, program mendeklarasikan dua class, yaitu Node untuk merepresentasikan setiap elemen dalam linked list dan DoublyLinkedList untuk mengelola linked list itu sendiri. Setiap node memiliki dua atribut, yaitu data (int) dan kata (string), serta dua pointer, prev dan next. Class DoublyLinkedList memiliki beberapa method, antara lain:

- a. push(int data, string kata): Menambahkan node baru ke depan linked list.
- b. pop(): Menghapus node dari depan linked list.
- c. update(int oldData, int newData, string oldKata, string newKata): Mengupdate nilai data dan kata pada node yang memenuhi kriteria tertentu.
- d. deleteAll(): Menghapus semua node dari linked list.
- e. display(): Menampilkan semua data dan kata dari setiap node dalam linked list.

Di dalam fungsi main, program menggunakan loop tak terbatas untuk menampilkan menu dan menerima input dari pengguna. Pengguna dapat memilih untuk menambahkan data, menghapus data, mengupdate data, menghapus semua data, menampilkan semua data, atau keluar dari program.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    // komponen/member
    int usia;
    string nama;
    Node *next;
};
Node *head;
Node *tail;
// Inisialisasi Node
void init()
{
    head = NULL;
    tail = NULL;
}
// Pengecekan
bool isEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}
// Tambah Depan
```

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

// Tambah Belakang
void insertBelakang(int nilai, string nama)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->nama = nama;
    baru->usia = nilai;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
    else
    {
        tail->next = baru;
    }
}
```



```

        tail = baru;
    }
}
// Hitung Jumlah List
int hitungList()
{
    Node *hitung;
    hitung = head;
    int jumlah = 0;
    while (hitung != NULL)
    {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}
// Tambah Tengah
void insertTengah(int usia, string nama, int posisi)
{
    if (posisi < 1 || posisi > hitungList() + 1)
    {
        cout << "Posisi diluar jangkauan" << endl;
    }
    else
    {
        Node *baru, *bantu;
        baru = new Node();

        baru->usia = usia;
        baru->nama = nama;

        if (posisi == 1)
        {
            insertDepan(usia, nama);

```

```

    }
    else if (posisi == hitungList() + 1)
    {
        insertBelakang(usia, nama);
    }
    else
    {
        // tranversing
        bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Depan
void hapusDepan()
{
    Node *hapus;
    if (isEmpty() == false)
    {
        if (head->next != NULL)
        {
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else
        {

```

```

        head = tail = NULL;

    }

}

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

}

// Hapus Belakang
void hapusBelakang()
{
    Node *hapus;
    Node *bantu;
    if (isEmpty() == false)
    {
        if (head != tail)
        {
            hapus = tail;
            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 Tengah
void hapusTengah(int posisi)
{
    Node *hapus, *bantu, *bantu2;
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi di luar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi)
        {
            if (nomor == posisi - 1)
            {
                bantu2 = bantu;
            }
            if (nomor == posisi)
            {
                hapus = bantu;
            }
            bantu = bantu->next;
            nomor++;
        }
        bantu2->next = bantu;
        delete hapus;
    }
}
```

```

    }
}

// Hapus berdasarkan nama
void hapusNama(string nama)
{
    Node *hapus, *bantu, *bantu2;
    if (isEmpty() == false)
    {
        bantu = head;
        bantu2 = head;
        while (bantu != NULL)
        {
            if (bantu->nama == nama)
            {
                if (bantu == head)
                {
                    hapusDepan();
                    break;
                }
                else if (bantu == tail)
                {
                    hapusBelakang();
                    break;
                }
                else
                {
                    bantu2->next = bantu->next;
                    delete bantu;
                    break;
                }
            }
            bantu2 = bantu;
            bantu = bantu->next;
        }
    }
}

```

```

    }
    if (bantu == NULL)
    {
        cout << "Data tidak ditemukan" << endl;
    }
}
else
{
    cout << "List masih kosong!" << endl;
}
}

// Ubah Depan
void ubahDepan(int usia, string nama)
{
    if (isEmpty() == false)
    {
        head->usia = usia;
        head->nama = nama;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah Tengah
void ubahTengah(int usia, string nama, int posisi)
{
    Node *bantu;
    if (isEmpty() == false)
    {
        if (posisi < 1 || posisi > hitungList())
        {
            cout << "Posisi di luar jangkauan" << endl;
        }
    }
}

```

```

        else if (posisi == 1)
        {
            cout << "Posisi bukan posisi tengah" << endl;
        }
        else
        {
            bantu = head;
            int nomor = 1;
            while (nomor < posisi)
            {
                bantu = bantu->next;
                nomor++;
            }
            bantu->usia = usia;
            bantu->nama = nama;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah Belakang
void ubahBelakang(int usia, string nama)
{
    if (isEmpty() == false)
    {
        tail->usia = usia;
        tail->nama = nama;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

```

```

}

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

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

```



```

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

// Main Program
int main()
{
    init();
    int pilihan, usia, posisi;
    string nama, namaLama;

    do
    {
        cout << "Menu:" << endl;
        cout << "1. Masukkan data" << endl;
        cout << "2. Tambah data" << endl;
        cout << "3. Tampilkan data" << endl;
        cout << "4. Ubah data" << endl;
        cout << "5. Hapus data" << endl;
        cout << "6. Keluar" << endl;
        cout << "Pilih menu: ";
        cin >> pilihan;

        switch (pilihan)
        {
            case 1:
                cout << "Masukkan nama: ";
                cin >> nama;
                cout << "Masukkan usia: ";
                cin >> usia;
                insertBelakang(usia, nama);

```

```

        break;
    case 2:
        cout << "Masukkan nama: ";
        cin >> nama;
        cout << "Masukkan usia: ";
        cin >> usia;
        cout << "Masukkan posisi: ";
        cin >> posisi;
        insertTengah(usia, nama, posisi);
        break;
    case 3:
        tampil();
        break;
    case 4:
        cout << "Masukkan nama lama: ";
        cin >> namaLama;
        cout << "Masukkan nama baru: ";
        cin >> nama;
        cout << "Masukkan usia baru: ";
        cin >> usia;
        // Cari posisi nama lama
        Node *bantu;
        bantu = head;
        posisi = 1;
        while (bantu != NULL)
        {
            if (bantu->nama == namaLama)
            {
                break;
            }
            bantu = bantu->next;
            posisi++;
        }
        ubahTengah(usia, nama, posisi);

```

```

        break;
    case 5:
        cout << "Masukkan nama yang akan dihapus: ";
        cin >> nama;
        hapusNama(nama);
        break;
    case 6:
        cout << "Keluar dari program..." << endl;
        break;
    default:
        cout << "Pilihan tidak valid" << endl;
    }

} while (pilihan != 6);

return 0;
}

```

Screenshoot program

a. Data Awal

```

Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
John 19
Jane 20
Michael 18
Yusuke 19
Akechi 20
Hoshino 18
Karin 18

```

b. Hapus Data Akechi

```
Masukkan usia: 18
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 5
Masukkan nama yang akan dihapus: Akechi
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
John 19
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18
```

c. Tambahkan data Futaba 18 diantara John dan Jane

```
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 2
Masukkan nama: Futaba
Masukkan usia: 18
Masukkan posisi: 2
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
John 19
Futaba 18
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18
```

d. Tambahkan data Igor 20 diawal

```
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 2
Masukkan nama: Igor
Masukkan usia: 20
Masukkan posisi: 1
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
Igor 20
John 19
Futaba 18
Jane 20
Michael 18
Yusuke 19
Hoshino 18
Karin 18
```

e. Ubah data Michael menjadi Reyn 18

```
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 4
Masukkan nama lama: Michael
Masukkan nama baru: Reyn
Masukkan usia baru: 18
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
Igor 20
John 19
Futaba 18
Jane 20
Reyn 18
Yusuke 19
Hoshino 18
Karin 18
```

- f. Tampilkan seluruh data

```
Menu:
1. Masukkan data
2. Tambah data
3. Tampilkan data
4. Ubah data
5. Hapus data
6. Keluar
Pilih menu: 3
Nama Usia
Igor 20
John 19
Futaba 18
Jane 20
Reyn 18
Yusuke 19
Hoshino 18
Karin 18
```

Deskripsi program

Program ini merupakan implementasi dari sebuah linked list non-circular. Pada program ini, setiap node memiliki dua data yaitu usia dan nama, serta pointer yang menunjukkan ke node selanjutnya dalam linked list.

Alur program dimulai dengan inisialisasi linked list (fungsi init) yang mengatur head dan tail menjadi NULL. Selanjutnya, terdapat fungsi-fungsi seperti isEmpty untuk memeriksa apakah linked list kosong, insertDepan dan insertBelakang untuk menambahkan node di depan dan di belakang, hitungList untuk menghitung jumlah node dalam linked list, serta fungsi-fungsi lain untuk mengubah, menghapus, dan menampilkan data dalam linked list.

Pada bagian main, program memberikan beberapa pilihan menu kepada pengguna seperti memasukkan data baru, menambah data pada posisi

tertentu, menampilkan data, mengubah data, menghapus data, dan keluar dari program. Setiap pilihan menu akan memanggil fungsi-fungsi yang sesuai untuk melakukan operasi pada linked list. Program akan terus berjalan sampai pengguna memilih untuk keluar dari program.

2. Unguided 2

Source code

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

class Node
{
public:
    string namaProduk;
    int harga;
    Node *prev;
    Node *next;
};

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

    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }

    void push(string namaProduk, int harga)
```

```
{  
    Node *newNode = new Node;  
    newNode->namaProduk = namaProduk;  
    newNode->harga = harga;  
    newNode->prev = nullptr;  
    newNode->next = head;  
  
    if (head != nullptr)  
    {  
        head->prev = newNode;  
    }  
    else  
    {  
        tail = newNode;  
    }  
  
    head = newNode;  
}  
  
void pushCenter(string namaProduk, int harga, int posisi)  
{  
    if (posisi < 0)  
    {  
        cout << "Posisi harus bernilai non-negatif." << endl;  
        return;  
    }  
  
    Node *newNode = new Node;  
    newNode->namaProduk = namaProduk;  
    newNode->harga = harga;  
  
    if (posisi == 0 || head == nullptr)  
    {  
        newNode->prev = nullptr;
```

```
newNode->next = head;

if (head != nullptr)
{
    head->prev = newNode;
}
else
{
    tail = newNode;
}
head = newNode;
}
else
{
    Node *temp = head;
    int count = 0;
    while (temp != nullptr && count < posisi)
    {
        temp = temp->next;
        count++;
    }

    if (temp == nullptr)
    {
        newNode->prev = tail;
        newNode->next = nullptr;
        tail->next = newNode;
        tail = newNode;
    }
    else
    {
        newNode->prev = temp->prev;
        newNode->next = temp;
        temp->prev->next = newNode;
    }
}
```

```
        temp->prev = 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 popCenter(int posisi)
{
    if (head == nullptr)
    {
        cout << "List kosong. Tidak ada yang bisa dihapus."
<< endl;
        return;
    }
}
```

```
if (posisi < 0)
{
    cout << "Posisi harus bernilai non-negatif." << endl;
    return;
}

if (posisi == 0)
{
    Node *temp = head;
    head = head->next;

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

    delete temp;
}
else
{
    Node *temp = head;
    int count = 0;
    while (temp != nullptr && count < posisi)
    {
        temp = temp->next;
        count++;
    }

    if (temp == nullptr)
```

```

        {
            cout << "Posisi melebihi ukuran list. Tidak ada
yang dihapus." << endl;
            return;
        }

        if (temp == tail)
        {
            tail = tail->prev;
            tail->next = nullptr;
            delete temp;
        }
        else
        {
            temp->prev->next = temp->next;
            temp->next->prev = temp->prev;
            delete temp;
        }
    }
}

bool update(string oldNamaProduk, string newNamaProduk, int
newHarga)
{
    Node *current = head;

    while (current != nullptr)
    {
        if (current->namaProduk == oldNamaProduk)
        {
            current->namaProduk = newNamaProduk;
            current->harga = newHarga;
            return true;
        }
    }
}

```

```

        current = current->next;
    }
    return false;
}

bool updateCenter(string newNamaProduk, int newHarga, int
posisi)
{
    if (head == nullptr)
    {
        cout << "List kosong. Tidak ada yang dapat
diperbarui." << endl;
        return false;
    }

    if (posisi < 0)
    {
        cout << "Posisi harus bernilai non-negatif." << endl;
        return false;
    }

    Node *current = head;
    int count = 0;

    while (current != nullptr && count < posisi)
    {
        current = current->next;
        count++;
    }

    if (current == nullptr)
    {
        cout << "Posisi melebihi ukuran list. Tidak ada yang
diperbarui." << endl;

```

```

        return false;
    }

    current->namaProduk = newNamaProduk;
    current->harga = newHarga;
    return true;
}

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

void display()
{
    if (head == nullptr)
    {
        cout << "List kosong." << endl;
        return;
    }

    Node *current = head;

    cout << setw(37) << setfill('-') << "-" << setfill(' ')
<< endl;
    cout << "| " << setw(20) << left << "Nama Produk"

```



```

        << " | " << setw(10) << "Harga"
        << " |" << endl;
        cout << setw(37) << setfill('-') << "-" << setfill(' ')
<< endl;

        while (current != nullptr)
        {
            cout << "| " << setw(20) << left << current-
>namaProduk << " | " << setw(10) << current->harga << " |" <<
endl;

            current = current->next;
        }
        cout << setw(37) << setfill('-') << "-" << setfill(' ')
<< endl;
    }
};

int main()
{
    DoublyLinkedList list;
    int choice;
    cout << endl
<< "Skincare Abadi" << endl;
    do
    {
        cout << "1. Tambah data" << endl;
        cout << "2. Hapus data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
        cout << "5. Hapus Data Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data" << endl;
        cout << "7. Tampilkan data" << endl;
        cout << "8. Exit" << endl;
    }
}

```

```

cout << "Pilihan : ";
cin >> choice;

switch (choice)
{
case 1:
{
    string namaProduk;
    int 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 newNamaProduk;
    int newHarga, posisi;
    cout << "Masukkan posisi produk: ";
    cin >> posisi;
    cout << "Masukkan nama baru produk: ";
    cin >> newNamaProduk;
    cout << "Masukkan harga baru produk: ";
    cin >> newHarga;

    bool updatedCenter =
list.updateCenter(newNamaProduk, newHarga, posisi);

```

```
        if (!updatedCenter)
        {
            cout << "Data not found" << endl;
        }
        break;
    }
    case 4:
    {
        string namaProduk;
        int harga, posisi;
        cout << "Masukkan posisi data produk: ";
        cin >> posisi;
        cout << "Masukkan nama produk: ";
        cin.ignore();
        getline(cin, namaProduk);
        cout << "Masukkan harga produk: ";
        cin >> harga;
        list.pushCenter(namaProduk, harga, posisi);
        break;
    }
    case 5:
    {
        int posisi;
        cout << "Masukkan posisi data produk: ";
        cin >> posisi;
        list.popCenter(posisi);
        break;
    }
    case 6:
    {
        list.deleteAll();
        break;
    }
    case 7:
```

```
{  
    list.display();  
    break;  
}  
case 8:  
{  
    return 0;  
}  
default:  
{  
    cout << "Invalid choice" << endl;  
    break;  
}  
}  
} while (choice != 8);  
  
return 0;  
}
```

Screenshoot program

- a. Tambahkan produk Azarine dengan harga 65000 diantara Somethinc dan Skintific

1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit

Pilihan : 4

Masukkan posisi data produk: 2

Masukkan nama produk: Azarine

Masukkan harga produk: 65000

1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit

Pilihan : 7

Nama Produk	Harga

Originote	60000
Somethinc	150000
Azarine	65000
Skintific	100000
Wardah	50000
Hanasui	30000

b. Hapus produk wardah

1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit

Pilihan : 5

Masukkan posisi data produk: 4

1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit

Pilihan : 7

Nama Produk	Harga

Originote	60000
Somethinc	150000
Azarine	65000
Skintific	100000
Hanasui	30000

- c. Update produk Hanasui menjadi Cleora dengan harga 55.000

```

1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit
Pilihan : 3
Masukkan posisi produk: 4
Masukkan nama baru produk: Cleora
Masukkan harga baru produk: 55000
1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit
Pilihan : 7
-----
| Nama Produk          | Harga      |
-----
| Originote            | 60000      |
| Somethinc             | 150000     |
| Azarine               | 65000      |
| Skintific             | 100000     |
| Cleora                | 55000      |
-----

```

d. Tampilan Akhir

```
1. Tambah data
2. Hapus data
3. Update data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan data
8. Exit
Pilihan : 7

-----
| Nama Produk      | Harga      |
-----
| Originote        | 60000      |
| Somethinc         | 150000     |
| Azarine           | 65000      |
| Skintific         | 100000     |
| Cleora            | 55000      |
-----
```

Deskripsi program

Program ini merupakan simulasi sederhana untuk mengelola data produk skincare. Program ini menggunakan struktur data Doubly Linked List untuk menyimpan informasi tentang nama produk dan harganya. Berikut adalah alur cerita dari program ini:

- a. Program dimulai dengan menampilkan menu utama yang berisi pilihan untuk melakukan operasi-operasi pada data produk skincare.
- b. Pengguna dapat memilih untuk menambah data produk, menghapus data produk, memperbarui data produk, menambah data produk pada posisi tertentu, menghapus data produk pada posisi tertentu, menghapus semua data, atau menampilkan semua data produk yang tersimpan.

- c. Untuk menambah data produk, pengguna diminta untuk memasukkan nama produk dan harganya. Data ini kemudian disimpan dalam Doubly Linked List.
- d. Untuk menghapus data produk, program akan menghapus data produk paling baru yang dimasukkan (pop).
- e. Untuk memperbarui data produk, pengguna diminta untuk memasukkan posisi data yang ingin diperbarui, nama baru produk, dan harga baru produk. Jika data ditemukan, data produk akan diperbarui.
- f. Untuk menambah data produk pada posisi tertentu, pengguna diminta untuk memasukkan posisi data, nama produk, dan harganya. Data produk akan ditambahkan pada posisi yang diminta.
- g. Untuk menghapus data produk pada posisi tertentu, pengguna diminta untuk memasukkan posisi data yang ingin dihapus. Data pada posisi tersebut akan dihapus dari Doubly Linked List.
- h. Untuk menghapus semua data produk, program akan menghapus semua data yang tersimpan dalam Doubly Linked List.
- i. Untuk menampilkan semua data produk yang tersimpan, program akan menampilkan nama produk dan harganya dalam tabel.
- j. Program akan terus berjalan dan menampilkan menu utama hingga pengguna memilih untuk keluar (pilihan 8).

BAB IV

KESIMPULAN

Single linked list adalah struktur data yang terdiri dari simpul-simpul yang saling terhubung, dimana setiap simpul memiliki dua bagian yaitu data dan pointer yang menunjuk ke simpul berikutnya. Dalam modul ini, program-program sederhana telah dibuat untuk mengilustrasikan cara kerja dasar dari single linked list, seperti penambahan data di depan, belakang, dan tengah, penghapusan data di depan, belakang, dan tengah, serta pengubahan data di depan, belakang, dan tengah. Program-program tersebut memperlihatkan konsep pengelolaan data dalam single linked list.

Double linked list adalah struktur data yang mirip dengan single linked list, namun setiap simpul memiliki dua pointer yaitu pointer yang menunjuk ke simpul sebelumnya dan pointer yang menunjuk ke simpul berikutnya. Dalam modul double linked list, program-program telah dibuat dengan fitur tambah, hapus, update, clear, display data, dan exit. Keunggulan dari double linked list terletak pada kemampuannya dalam akses data secara maju dan mundur serta fleksibilitasnya dalam pengelolaan data, sesuai dengan kebutuhan aplikasi yang memerlukan struktur data yang kompleks.