

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 3
“SINGLE AND DOUBLE LINKED LIST”**



**DISUSUN OLEH :
BINTANG YUDHISTIRA
2311102052**

**DOSEN
WAHYU ANDI SAPUTRA, S.PD., M.PD.**

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

A. Dasar Teori

Linked list adalah struktur data linier berbentuk rantai simpul di mana setiap simpul menyimpan 2 item, yaitu nilai data dan pointer ke simpul elemen berikutnya. Berbeda dengan array, elemen linked list tidak ditempatkan dalam alamat memori yang berdekatan melainkan elemen ditautkan menggunakan pointer.

Simpul pertama dari linked list disebut sebagai head atau simpul kepala. Apabila linked list berisi elemen kosong, maka nilai pointer dari head menunjuk ke NULL. Begitu juga untuk pointer berikutnya dari simpul terakhir atau simpul ekor akan menunjuk ke NULL.

Ukuran elemen dari linked list dapat bertambah secara dinamis dan mudah untuk menyisipkan dan menghapus elemen karena tidak seperti array, kita hanya perlu mengubah pointer elemen sebelumnya dan elemen berikutnya untuk menyisipkan atau menghapus elemen.

Linked list biasanya digunakan untuk membuat file system, adjacency list, dan hash table.

JENIS JENIS LINKEDLIST YANG AKAN DI BAHAS PADA MODUL INI :

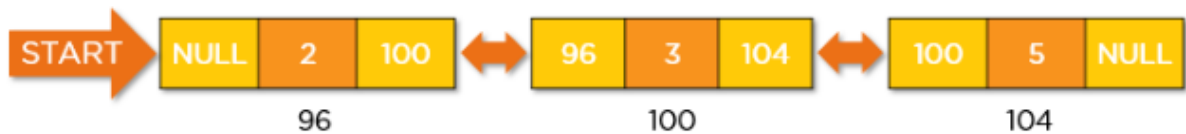
- **SINGLE LINKED LIST**

Singly linked list adalah linked list unidirectional. Jadi, kita hanya dapat melintasinya dalam satu arah, yaitu dari simpul kepala ke simpul ekor.



- **DOUBLE LINKED LIST**

Doubly linked list adalah linked list bidirectional. Jadi, kita bisa melintasinya secara dua arah. Tidak seperti singly linked list, simpul doubly linked list berisi satu pointer tambahan yang disebut previous pointer. Pointer ini menunjuk ke simpul sebelumnya.



KARAKTERISTIK LINKED LIST

- Linked list menggunakan memori tambahan untuk menyimpan link (tautan)
- Untuk inisialisasi awal linked list, kita tidak perlu tahu ukuran dari elemen.
- Linked list umumnya dapat digunakan untuk mengimplementasikan struktur data lain seperti stack, queue, ataupun graf
- Simpul pertama dari linked list disebut sebagai Head.
- Pointer setelah simpul terakhir selalu bernilai NULL
- Dalam struktur data linked list, operasi penyisipan dan penghapusan dapat dilakukan dengan mudah
- Tiap-tiap simpul dari linked list berisi pointer atau tautan yang menjadi alamat dari simpul berikutnya
- Linked list bisa menyusut atau bertambah kapan saja dengan mudah.

FUNGSI DAN KEGUNAAN LINKED LIST

- ❖ Linked list dapat digunakan untuk mengimplementasikan struktur data lain seperti stack, queue, graf, dll.
- ❖ Digunakan untuk melakukan operasi aritmatika pada bilangan long integer
- ❖ Dipakai untuk representasi matriks rongga.
- ❖ Digunakan dalam alokasi file yang ditautkan.
- ❖ Membantu dalam manajemen memori.

B. Guided

Guided 1 : Latihan Single Linked List

Source Code :

```
#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;
    }
}

```

```

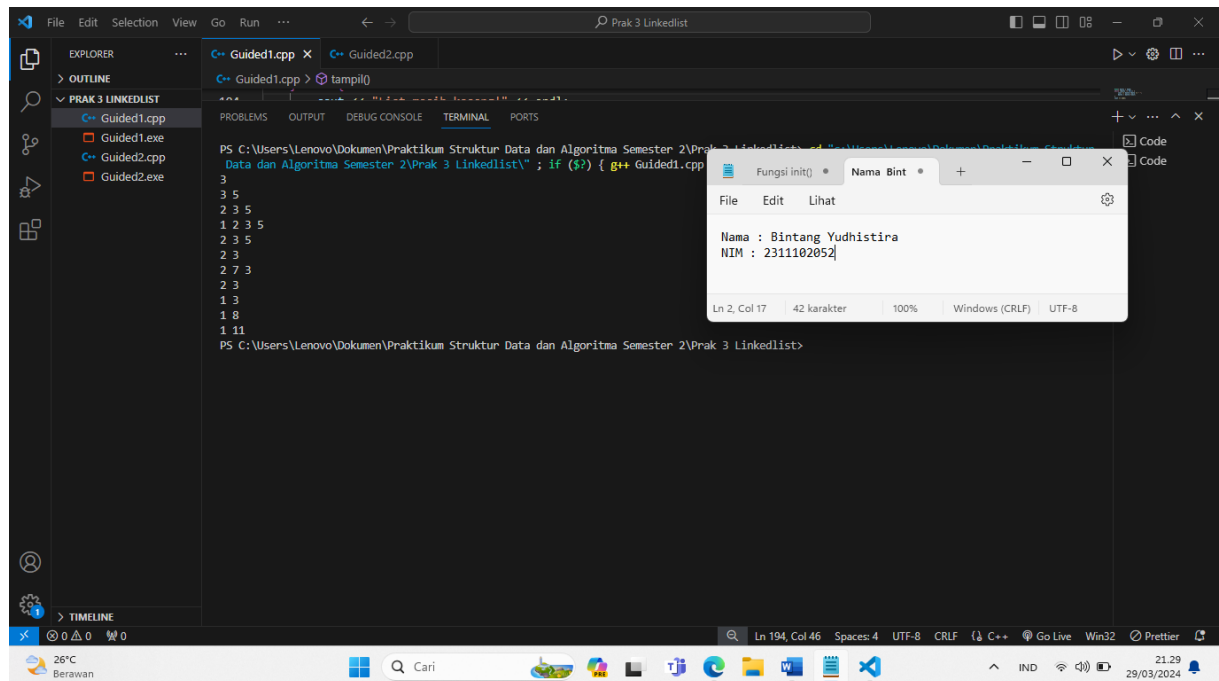
// Hapus semua Node di list
void clearList() {
    Node* bantu = head;
    while (bantu != NULL) {
        Node* hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}

// 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;
}

```


Screenshots Output:



```
PS C:\Users\Lenovo\Dokumen\Praktikum Struktur Data dan Algoritma Semester 2\Prak 3 LinkedList> g++ Guided1.cpp
PS C:\Users\Lenovo\Dokumen\Praktikum Struktur Data dan Algoritma Semester 2\Prak 3 LinkedList> if ($?) { g++ Guided1.cpp
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 C:\Users\Lenovo\Dokumen\Praktikum Struktur Data dan Algoritma Semester 2\Prak 3 LinkedList>
```

Nama : Bintang Yudhistira
NIM : 2311102052

Deskripsi Program :

Fungsi `init()` bertanggung jawab untuk menginisialisasi pointer `head` dan `tail` menjadi `NULL`, menandakan bahwa linked list kosong. Fungsi `isEmpty()` digunakan untuk memeriksa apakah linked list kosong atau tidak. Kemudian, fungsi `insertDepan(int nilai)` dan `insertBelakang(int nilai)` menambahkan node baru di depan dan belakang linked list, berturut-turut. Fungsi `hitungList()` menghitung jumlah node dalam linked list. Fungsi `insertTengah(int data, int posisi)` menambahkan node baru di posisi tengah linked list. Sementara itu, fungsi `hapusDepan()` dan `hapusBelakang()` menghapus node pertama dan terakhir dari linked list.

Untuk mengubah nilai data dari node, ada fungsi `ubahDepan(int data)`, `ubahTengah(int data, int posisi)`, dan `ubahBelakang(int data)`. Fungsi-fungsi tersebut mengubah nilai data dari node pertama, node di posisi tengah, dan node terakhir linked list. Terakhir, fungsi `clearList()` menghapus semua node dari linked list, sementara `tampil()` menampilkan semua nilai data dalam linked list. Fungsi `main()` digunakan untuk menguji implementasi linked list dengan memanggil fungsi-fungsi tersebut.

Guided 2 : Latihan Double Linked List

Source Code :

```
#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: {

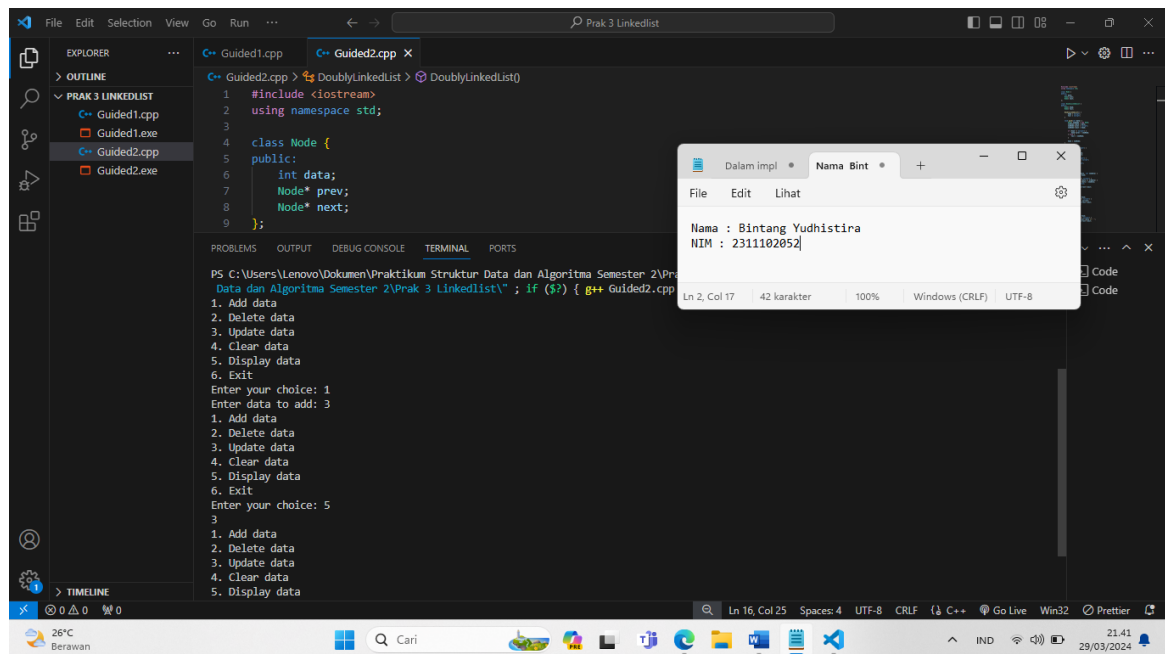
```

```

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

```

Screenshots Output:



Deskripsi Program :

Dalam implementasi ini, terdapat dua kelas utama: kelas `Node`, yang mewakili setiap node dalam linked list, dan kelas `DoublyLinkedList`, yang merupakan struktur data linked list itu sendiri. Kelas `Node` memiliki tiga atribut: `data` untuk menyimpan nilai dari node, `prev` sebagai pointer ke node sebelumnya, dan `next` sebagai pointer ke node berikutnya. Kelas `DoublyLinkedList` memiliki dua pointer utama: `head` yang menunjuk ke node pertama dalam linked list, dan `tail` yang menunjuk ke node terakhir dalam linked list. Konstruktor kelas `DoublyLinkedList` menginisialisasi `head` dan `tail` menjadi `nullptr`, menandakan bahwa linked list awalnya kosong. Dalam kelas `DoublyLinkedList`, terdapat beberapa metode penting. Metode `push(int data)` digunakan untuk menambahkan node baru di depan linked list dengan nilai `data`. Metode `pop()`

menghapus node pertama dari linked list. Metode `update(int oldData, int newData)` mencari node dengan nilai `oldData` dan menggantinya dengan `newData`. Metode `deleteAll()` menghapus semua node dari linked list, sementara metode `display()` menampilkan semua nilai data dalam linked list. Fungsi `main()` digunakan untuk menguji implementasi linked list dengan memberikan opsi kepada pengguna untuk menambah, menghapus, mengubah, menampilkan, atau membersihkan data dalam linked list. Program akan berjalan terus menerima input dari pengguna hingga pengguna memilih untuk keluar dari program..

C. Unguided

1). Soal mengenai Single Linked List

Buatlah program menu Single Linked List Non-Circular untuk menyimpan Nama dan usia mahasiswa, dengan menggunakan inputan dari user. Lakukan operasi berikut:

a. Masukkan data sesuai urutan berikut. (Gunakan insert depan, belakang atau tengah). Data pertama yang dimasukkan adalah nama dan usia anda.

[Nama_anda] [Usia_anda]

John 19

Jane 20

Michael 18

Yusuke 19

Akechi 20

Hoshino 18

Karin 18

- b. Hapus data Akechi
- c. Tambahkan data berikut diantara John dan Jane : Futaba 18
- d. Tambahkan data berikut diawal : Igor 20
- e. Ubah data Michael menjadi : Reyn 18
- f. Tampilkan seluruh data

Source Code:

```
// UNGUIDED 1
#include <iostream>
using namespace std;

// Deklarasi Struct Node
struct Node
{
    string nama;
    int usia;
    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(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;
    }
}

// Tambah Node di belakang
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;
    }
}

// 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(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;
}
}

```

// 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(string nama, int usia)

```

```

{
    if (!isEmpty())
    {
        head->nama = nama;
        head->usia = usia;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di posisi tengah
void ubahTengah(string nama, int usia, 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->nama = nama;
            bantu->usia = usia;
        }
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di belakang
void ubahBelakang(string nama, int usia)
{
    if (!isEmpty())
    {
        tail->nama = nama;
        tail->usia = usia;
    }
}

```

```

    }
    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->nama << " ";
            cout << bantu->usia << " , ";
            bantu = bantu->next;
        }
        cout << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

int main()
{
    init();
    int menu, usia, posisi;
    string nama;
    cout << "\n# Menu Linked List Mahasiswa #" << endl;
    do
    {

```

```
cout << "\n 1. Insert Depan"
      << "\n 2. Insert Belakang"
      << "\n 3. Insert Tengah"
      << "\n 4. Hapus Depan"
      << "\n 5. Hapus Belakang"
      << "\n 6. Hapus Tengah"
      << "\n 7. Ubah Depan"
      << "\n 8. Ubah Belakang"
      << "\n 9. Ubah Tengah"
      << "\n 10. Tampilkan"
      << "\n 0. Keluar Program"
      << "\n Pilihan : ";

cin >> menu;
switch (menu)
{
case 1:
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    insertDepan(nama, usia);
    cout << endl;
    tampil();
    break;
case 2:
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    insertBelakang(nama, usia);
    cout << endl;
    tampil();
    break;
case 3:
    cout << "Masukkan Posisi : ";
    cin >> posisi;
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    insertTengah(nama, usia, posisi);
    cout << endl;
    tampil();
    break;
case 4:
    hapusDepan();
    cout << endl;
    tampil();
    break;
```

```

case 5:
    hapusBelakang();
    cout << endl;
    tampil();
    break;
case 6:
    cout << "Masukkan Posisi : ";
    cin >> posisi;
    hapusTengah(posisi);
    cout << endl;
    tampil();
    break;
case 7:
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    ubahDepan(nama, usia);
    cout << endl;
    tampil();
    break;
case 8:
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    ubahBelakang(nama, usia);
    cout << endl;
    tampil();
    break;
case 9:
    cout << "Masukkan Posisi : ";
    cin >> posisi;
    cout << "Masukkan Nama : ";
    cin >> nama;
    cout << "Masukkan Usia : ";
    cin >> usia;
    ubahTengah(nama, usia, posisi);
    cout << endl;
    tampil();
    break;
case 10:
    tampil();
    break;

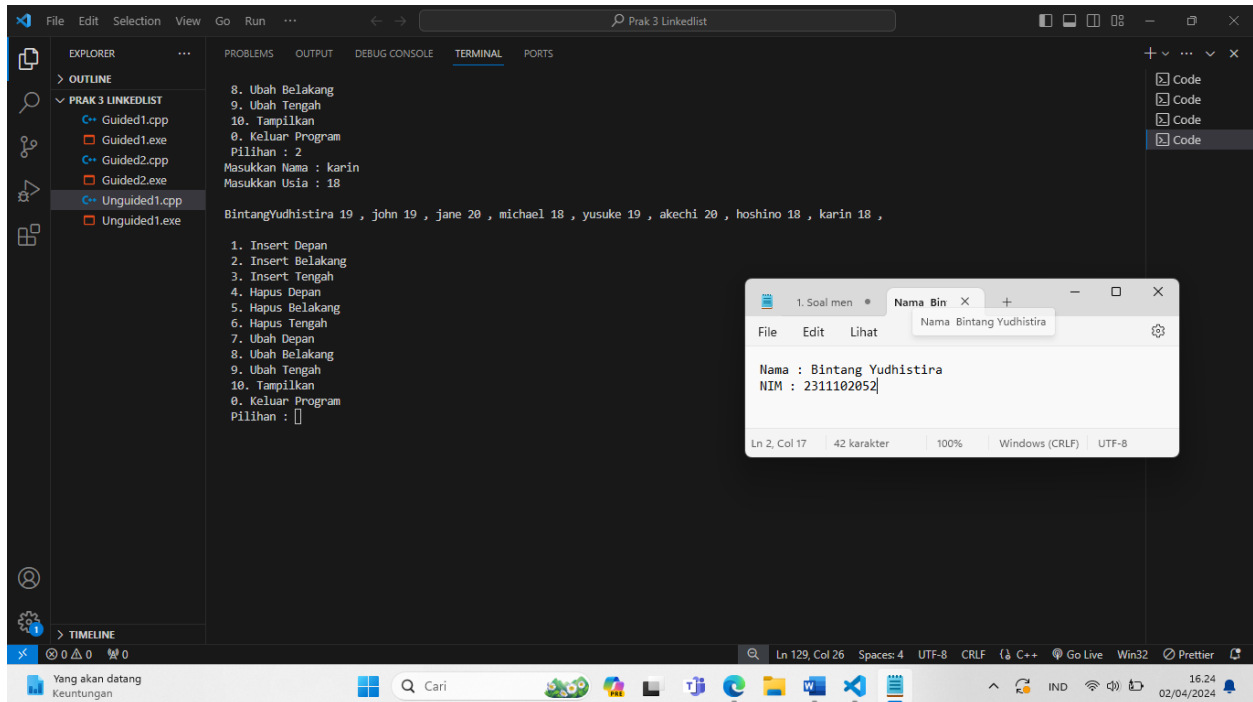
default:
    cout << "Pilihan Salah" << endl;
    break;
}

```

```
} while (menu != 0);  
return 0;  
}
```

Screenshot output:

a. Masukan Data



The screenshot shows a Windows desktop with two open applications. The primary application is Visual Studio Code, which is open to a C++ project named 'PRAK 3 LINKEDLIST'. The 'TERMINAL' tab is active, displaying the output of a compiled program. The program prompts the user to choose an option (0-10) to perform an operation on a linked list. The user has entered '0' to 'Keluarkan Program' (Exit Program). The terminal also shows the user's input for a new node: 'Masukkan Posisi : 6', 'Masukkan Nama : futaba', and 'Masukkan Usia : 18'. The program then displays the updated linked list: 'BintangYudhistira 19, john 19, jane 20, michael 18, yusuke 19, hoshino 18, karin 18, BintangYudhistira 19, john 19, futaba 18, jane 20, michael 18, yusuke 19, hoshino 18, karin 18'. A secondary application, Notepad, is open in the foreground, showing a text file named 'Nama Bint' containing the text 'Nama : Bintang Yudhistira' and 'NIM : 2311102052'.

Visual Studio Code Interface:

- Explorer:** Shows the project structure with files 'Guided1.cpp', 'Guided1.exe', 'Guided2.cpp', 'Guided2.exe', 'Unguided1.cpp', and 'Unguided1.exe'.
- Terminal:**

```

0. Keluar Program
Pilihan : 6
Masukkan Posisi : 6

BintangYudhistira 19, john 19, jane 20, michael 18, yusuke 19, hoshino 18, karin 18 ,

1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 3
Masukkan Posisi : 3
Masukkan Nama : futaba
Masukkan Usia : 18

BintangYudhistira 19, john 19, futaba 18, jane 20, michael 18, yusuke 19, hoshino 18, karin 18 ,

1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 0

```

Notepad Window:

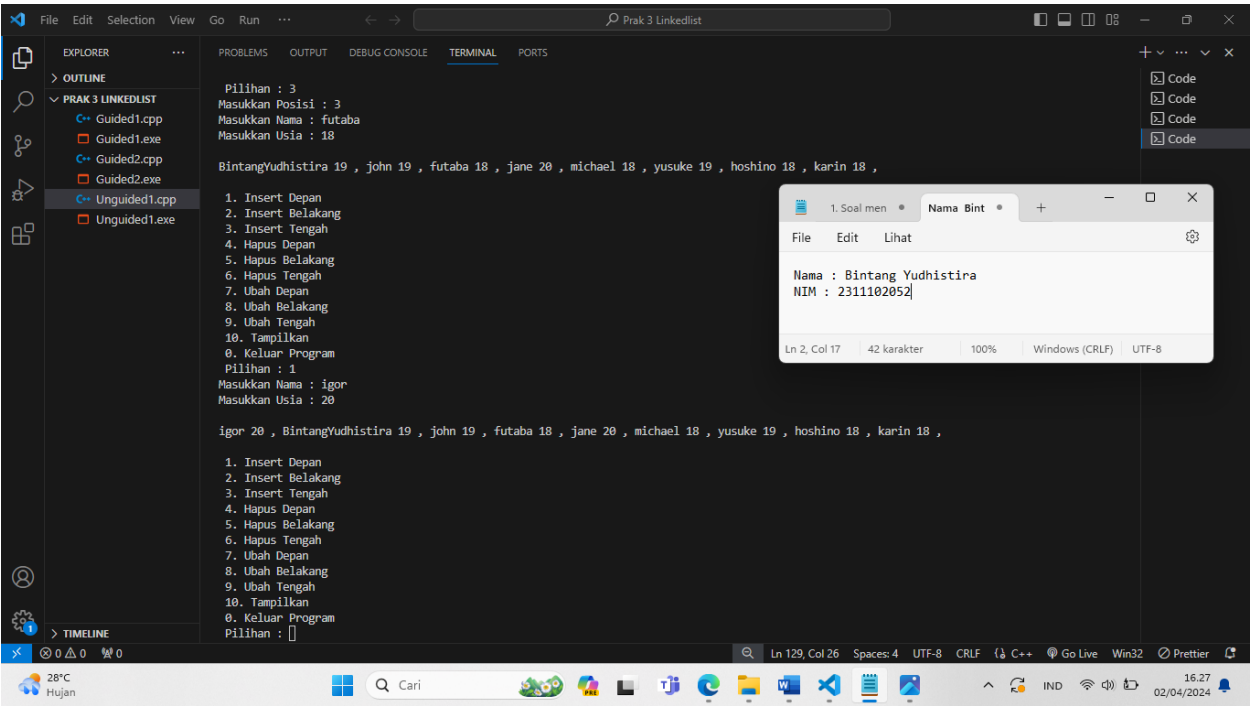
- Title Bar:** 1. Soal men * Nama Bint *
- Menu Bar:** File Edit Lihat
- Text Content:**

```

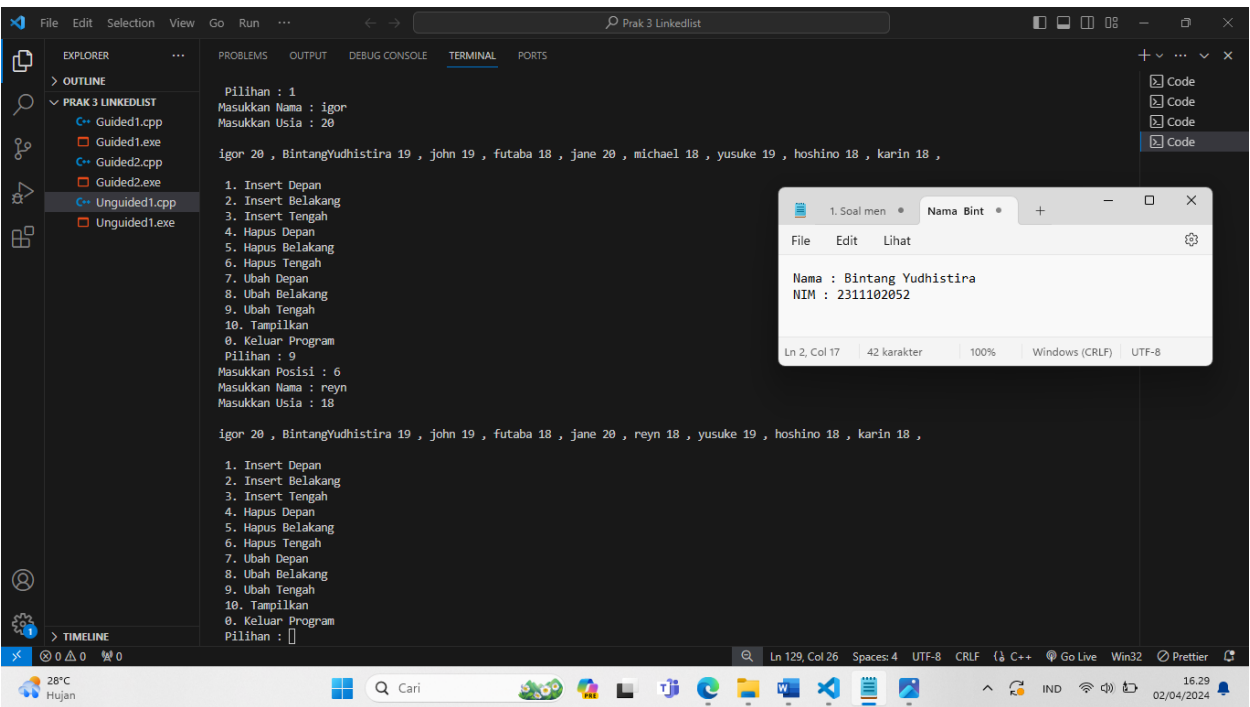
Nama : Bintang Yudhistira
NIM : 2311102052

```
- Status Bar:** Ln 2, Col 17 | 42 karakter | 100% | Windows (CRLF) | UTF-8

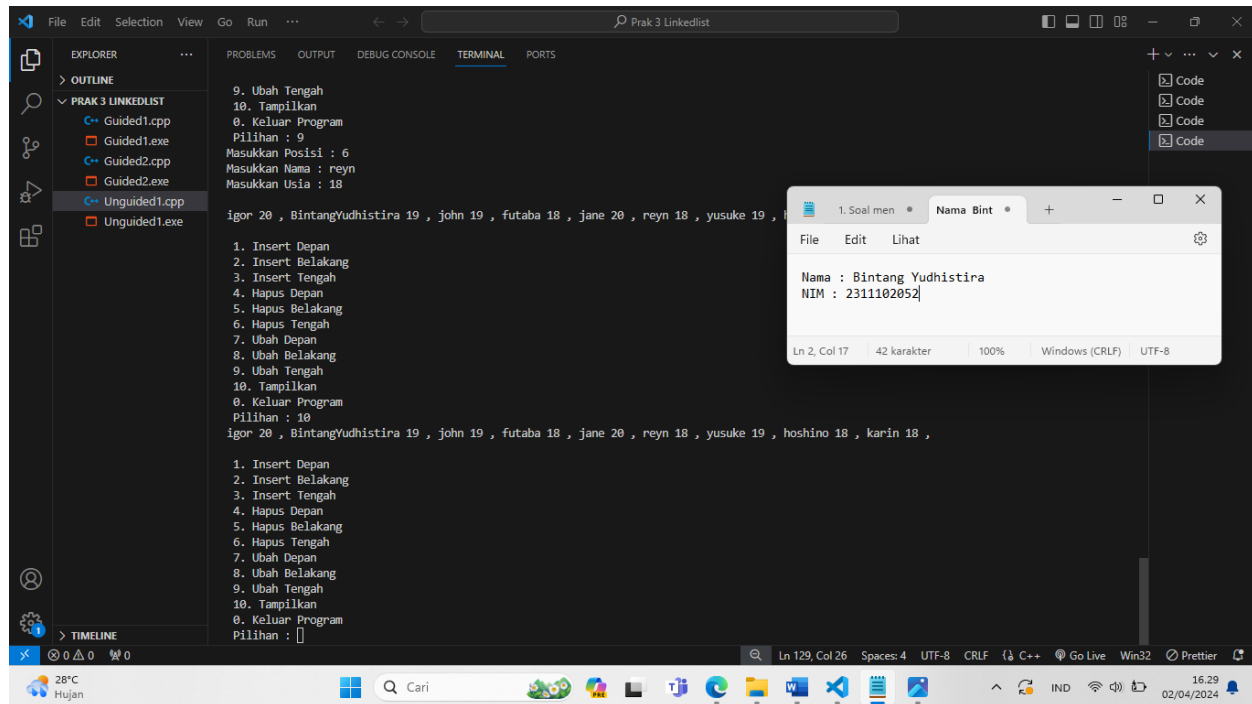
d. Tambahkan data berikut diawal : Igor 20



e. Ubah data Michael menjadi : Reyn 18



f. Tampilkan seluruh data



Deskripsi Program :

Setiap elemen (node) memiliki dua bagian: satu berisi data dan yang lainnya berisi alamat (pointer) ke node berikutnya.

Berikut adalah penjelasan singkat untuk setiap bagian dari kode:

- Deklarasi Struct Node: Mendefinisikan struktur Node yang berisi data nama (string), usia (integer), dan pointer next ke node berikutnya dalam linked list.
- Variabel Global head dan tail: Variabel global yang menunjukkan kepala (head) dan ekor (tail) dari linked list.
- Fungsi init(): Menginisialisasi linked list dengan mengatur head dan tail menjadi NULL.
- Fungsi isEmpty(): Mengembalikan true jika linked list kosong (head adalah NULL), dan false jika tidak.
- Fungsi insertDepan(): Menambahkan node baru di depan linked list.
- Fungsi insertBelakang(): Menambahkan node baru di belakang linked list.
- Fungsi hitungList(): Menghitung jumlah node dalam linked list.

- Fungsi insertTengah(): Menambahkan node baru di posisi tengah linked list.
- Fungsi hapusDepan(): Menghapus node pertama dari linked list.
- Fungsi hapusBelakang(): Menghapus node terakhir dari linked list.
- Fungsi hapusTengah(): Menghapus node di posisi tengah linked list.
- Fungsi ubahDepan(): Mengubah data dari node pertama.
- Fungsi ubahTengah(): Mengubah data dari node di posisi tengah.
- Fungsi ubahBelakang(): Mengubah data dari node terakhir.
- Fungsi clearList(): Menghapus semua node dari linked list.
- Fungsi tampil(): Menampilkan semua data node dalam linked list.
- Fungsi main(): Program utama yang memanfaatkan fungsi-fungsi di atas untuk melakukan manipulasi linked list seperti menyisipkan, menghapus, mengubah, dan menampilkan data node. Program berjalan dalam loop hingga pengguna memilih untuk keluar dari program.

2). Soal mengenai Double Linked List Modifikasi Guided Double Linked List dilakukan dengan penambahan operasi untuk menambah data, menghapus, dan update di tengah / di urutan tertentu yang diminta. Selain itu, buatlah agar tampilannya menampilkan Nama produk dan harga.

Nama Produk	Harga
Orginote	60.000
Somethinc	150.000
Skintific	100.000
Wardah	50.000
Hanasui	30.000

Case:

1. Tambahkan produk Azarine dengan harga 65000 diantara

Somethinc dan Skintific

2. Hapus produk wardah
3. Update produk Hanasui menjadi Cleora dengan harga 55.000
4. Tampilkan menu seperti dibawah ini

Toko Skincare Purwokerto

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

Pada menu 7, tampilan akhirnya akan menjadi seperti dibawah

Ini :

Nama Produk	Harga
Originote	60.000
Somethinc	150.000
Azarine	65.000
Skintific	100.000
Cleora	55.000

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
        << "Toko Skincare Purwokerto" << 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: ";

```

```

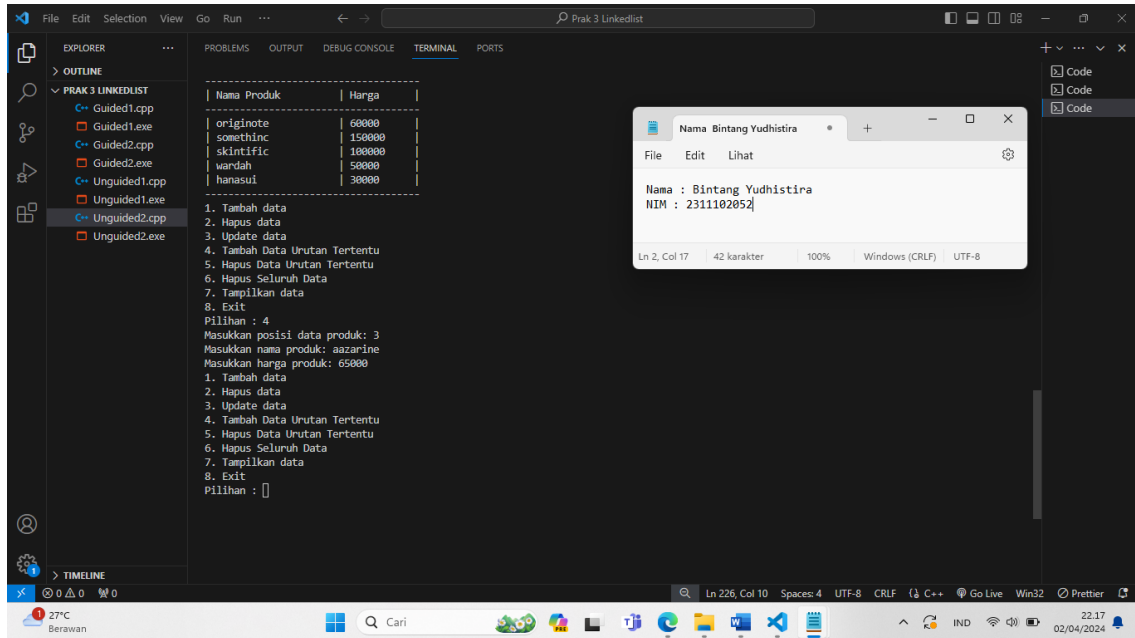
        cin >> harga;
        list.pushCenter(namaProduk, harga, posisi);
        break;
    }
    case 5:
    {
        int posisi;
        cout << "Masukkan posisi data produk: ";
        cin >> posisi;
        list.popCenter(posisi);
        break;
    }
    case 6:
    {
        list.deleteAll();
        break;
    }
    case 7:
    {
        list.display();
        break;
    }
    case 8:
    {
        return 0;
    }
    default:
    {
        cout << "Invalid choice" << endl;
        break;
    }
}
} while (choice != 8);

return 0;
}

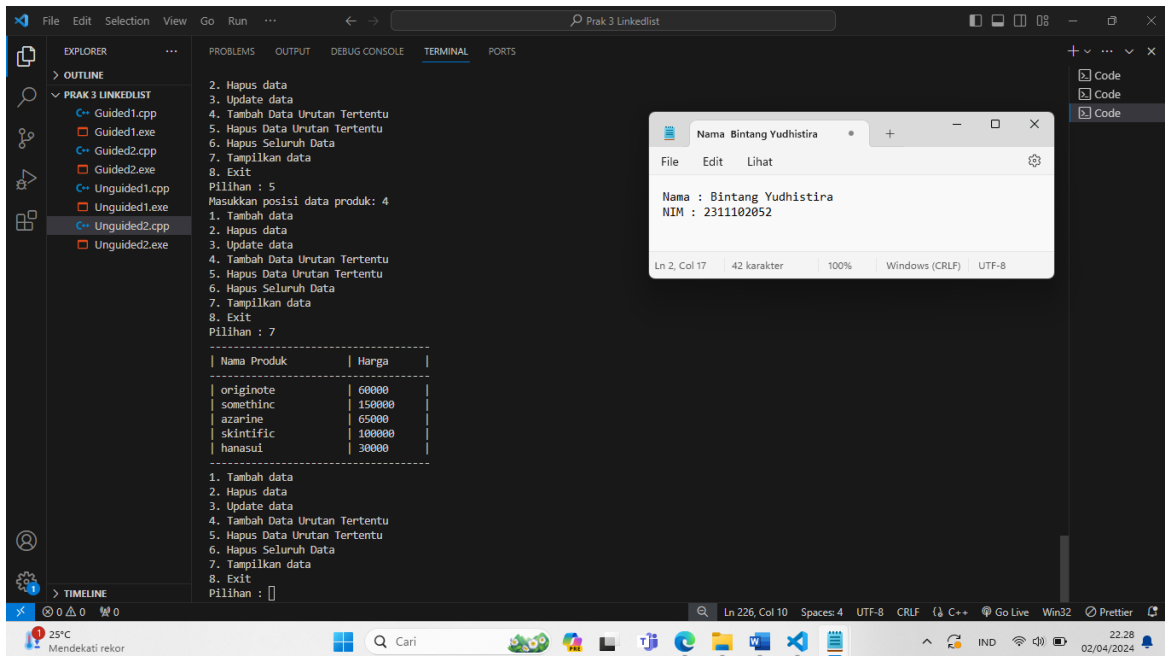
```

Screenshot Output:

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



b. Hapus produk wardah



c. Update produk Hanasui menjadi Cleora dengan harga 55.000

```
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

```
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 : []
```

Nama : Bintang Yudhistira
NIM : 2311102052

Deskripsi Program:

- Deklarasi Class Node: Mendefinisikan class Node yang berisi atribut namaProduk (string), harga (integer), dan pointer prev dan next ke node sebelumnya dan node selanjutnya.
- Deklarasi Class DoublyLinkedList: Mendefinisikan class DoublyLinkedList yang berisi atribut head dan tail yang merupakan pointer ke node pertama dan terakhir dalam doubly linked list.
- Metode push(): Menambahkan node baru di awal doubly linked list.
- Metode pushCenter(): Menambahkan node baru di posisi tertentu dalam doubly linked list.
- Metode pop(): Menghapus node dari awal doubly linked list.
- Metode popCenter(): Menghapus node dari posisi tertentu dalam doubly linked list.
- Metode update(): Mengupdate data node dengan nama produk tertentu dalam doubly linked list.
- Metode updateCenter(): Mengupdate data node pada posisi tertentu dalam doubly

linked list.

- Metode deleteAll(): Menghapus semua node dalam doubly linked list.
- Metode display(): Menampilkan semua data node dalam doubly linked list.
- Fungsi main(): Program utama yang menggunakan class DoublyLinkedList untuk melakukan manipulasi data dalam doubly linked list. Pengguna dapat menambahkan, menghapus, mengupdate, dan menampilkan data produk skincare dari toko "Toko Skincare Purwokerto".

D. Kesimpulan

Single linked list dan double linked list adalah dua jenis struktur data yang digunakan untuk menyimpan dan mengorganisir data dalam urutan linear. Single linked list memiliki setiap node yang terhubung ke node berikutnya melalui satu pointer, sementara double linked list memiliki setiap node yang terhubung ke node sebelumnya dan berikutnya melalui dua pointer. Dalam single linked list, operasi seperti penambahan dan penghapusan node cenderung lebih sederhana, tetapi operasi di posisi tengah memerlukan traversal dari awal. Di sisi lain, double linked list memberikan kemampuan traversal maju dan mundur dengan mudah, serta lebih efisien dalam operasi di posisi tengah karena memiliki pointer ke node sebelumnya. Keduanya memiliki keunggulan dan kelemahan masing-masing, dan pilihan antara keduanya tergantung pada kebutuhan spesifik dari aplikasi yang sedang dikembangkan.

E. Referensi

[1] Asisten Pratikum “Modul 3 Single dan Double Linkedlist”, Learning Management System, 2024.

[2] Educative. Singly linked list in C++. Diakses pada 31 Maret 2024.

Diakses Pada 2 April 2024, dari

<https://www.educative.io/answers/singly-linked-list-in-cpp>