

LAPORAN PRAKTIKUM

MODUL III “Single & Double Linked List”



Disusun oleh:

Shiva Indah Kurnia
NIM 2311102035

Dosen Pengampu:

Wahyu Andi Saputra, S.Pd., M.Eng

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

TUJUAN PRAKTIKUM

Setelah melakukan praktikum ini diharapkan mahasiswa dapat:

1. Mahasiswa diharapkan mampu untuk memahami perbedaan dari konsep single linked list dan double linked list.
2. Mahasiswa diharapkan mampu untuk mengaplikasikan konsep single linked list dan double linked list ke dalam pemrograman
3. Mahasiswa diharapkan mampu untuk menyelesaikan soal permasalahan yang melibatkan single linked list dan double linked list.

DASAR TEORI

Linked List adalah salah satu bentuk struktur data, berisi kumpulan data (node) yang tersusun secara sekuensial, saling sambung menyambung, dinamis dan terbatas. Linked List sering disebut juga Senarai Berantai. Linked List saling terhubung dengan bantuan variabel pointer. Masing-masing data dalam Linked List disebut dengan node (simpul) yang menempati alokasi memori secara dinamis dan biasanya berupa struct yang terdiri dari beberapa field.

- a) **Single Linked List** adalah sebuah Linked List yang menggunakan sebuah variabel pointer saja untuk menyimpan banyak data dengan metode Linked List, suatu daftar isi yang saling berhubungan. Jika dalam pointer Linked List individu hanya bisa berubah menjadi satu arah saja, maju/mundur, kanan/kiri, data pencarian saja hanya dalam satu arah. Single Linked List menggunakan *head* dan *tail*. Single Linked List ini membutuhkan dua variabel. *Head* dan *Tail*. Kepala selalu menunjuk pada simpul pertama while, Sedangkan kemauan ekor selalu menunjuk ke sebuah simpul terakhir. Perumpamaanya semua kotak adalah node, dua kotak node pertama (kotak pertama disebut next dan kotak kedua disebut next, dua kotak selanjutnya adalah data dan next, sedangkan dua kotak terakhir sebelum kotak terakhir adalah data dan kotak paling terakhir adalah null, kotak terakhir disebut tail.
- b) **Double Linked List** adalah suatu linked list yang mempunyai 2 penunjuk yaitu penunjuk ke simpul sebelumnya dan ke simpul berikutnya. Setiap node dalam double linked list memiliki dua pointer yaitu pointer ke node sebelumnya (prev) dan pointer ke node berikutnya (next). Double linked list memungkinkan untuk mengakses node sebelumnya maupun node berikutnya dari suatu node. Setiap double linked list memiliki dua pointer yaitu pointer ke head (awal) dan pointer ke tail (akhir) dari linked list tersebut..

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->data = nilai;
    baru->kata = kata;
    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->data = nilai;
    baru->kata = kata;
    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;
        }
    }
    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;
}

```

```

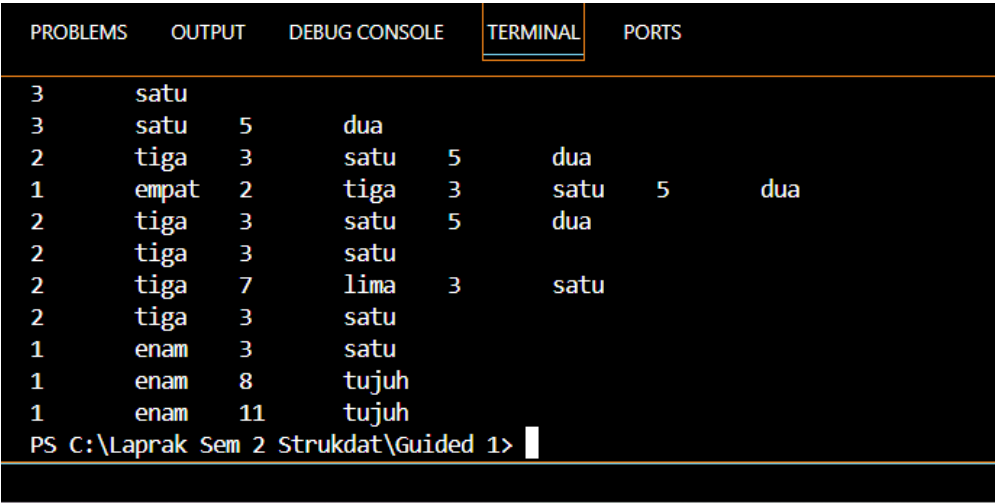
// Tampilkan List
void tampil()
{
    Node *bantu;
    bantu = head;
    if (isEmpty() == false)
    {
        while (bantu != NULL)
        {
            cout << bantu->data << "\t";
            cout << bantu->kata<<"\t";
            bantu = bantu->next;
        }
        cout << endl;
    }
    else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

int main()
{
    init();
    insertDepan(3, "satu");
    tampil();
    insertBelakang(5, "dua");
    tampil();
    insertDepan(2, "tiga");
    tampil();
    insertDepan(1, "empat");
    tampil();
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7,"lima", 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1,"enam") ;
    tampil();
    ubahBelakang(8,"tujuh");
    tampil();
    ubahTengah(11,"delapan", 2);
    tampil();
}

```

```
return 0;
}
```

Screenshot Program



Deskripsi Program

Ini adalah program C++ yang mengimplementasikan struktur data linked list dengan satu arah (singly linked list). Program ini menyediakan beberapa operasi untuk memanipulasi linked list, seperti menambahkan node baru ke awal, tengah, atau akhir dari daftar, menghapus node dari awal, tengah, atau akhir daftar, memperbarui data node, dan menampilkan isi dari daftar.

Program dimulai dengan mendeklarasikan struktur Node yang memiliki dua anggota: data, sebuah bilangan bulat yang menyimpan nilai node, dan next, sebuah pointer ke node berikutnya dalam daftar. Program juga mendeklarasikan dua pointer global ke Node, head dan tail, yang menunjuk pada node pertama dan terakhir dalam daftar, masing-masing.

Program kemudian mendefinisikan beberapa fungsi untuk memanipulasi linked list. Fungsi-fungsi ini termasuk init(), yang menginisialisasi daftar dengan mengatur head dan tail menjadi NULL, isEmpty(), yang memeriksa apakah daftar kosong dengan menguji apakah head adalah NULL, insertDepan(), yang memasukkan node baru di awal daftar, insertBelakang(), yang memasukkan node baru di akhir daftar, hitungList(), yang menghitung jumlah node dalam daftar, insertTengah(), yang memasukkan node baru pada posisi tertentu dalam daftar,

hapusDepan(), yang menghapus node pertama dalam daftar, hapusBelakang(), yang menghapus node terakhir dalam daftar, hapusTengah(), yang menghapus node pada posisi tertentu dalam daftar, ubahDepan(), yang memperbarui nilai dari node pertama dalam daftar, ubahTengah(), yang memperbarui nilai dari node pada posisi tertentu dalam daftar, ubahBelakang(), yang memperbarui nilai dari node terakhir dalam daftar, dan clearList(), yang menghapus semua node dalam daftar. Program juga mendefinisikan fungsi tampil() yang menampilkan isi daftar dengan menelusuri daftar menggunakan pointer Node, bantu, yang dimulai dari head daftar dan bergerak dari node ke node sampai mencapai akhir daftar

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 newKata) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == oldData) {
            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 << endl;
            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 newKata;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                cout << "Enter new kata: ";

```

```

        cin >> newKata;
        bool updated = list.update(oldData,
        newData, 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

```

2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 35
Enter kata to add: Shiva
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
35 Shiva

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

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS C:\Laparak Sem 2 Struktad\Guided 1>

```

Deskripsi Program

Program di atas adalah sebuah implementasi dari struktur data doubly linked list dengan bahasa pemrograman C++. Program ini menyediakan beberapa operasi seperti menambah, menghapus, mengupdate dan menampilkan data pada linked list. Program ini memanfaatkan dua class yaitu class Node dan class DoublyLinkedList. Class Node digunakan untuk membuat sebuah node yang memiliki atribut data, pointer prev, dan pointer next. Sedangkan class DoublyLinkedList digunakan untuk membuat linked list yang terdiri dari beberapa operasi seperti push, pop, update, deleteAll, dan display. Pada program utama, pengguna dapat memilih operasi yang ingin dilakukan pada linked list dengan memasukkan input berupa angka pada menu yang telah disediakan.

UNGUIDED

1. Unguided 1

Source Code

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

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

class SingleLinkedList {
private:
    Node* head;
public:
    SingleLinkedList() {
        head = NULL;
    }
    // Insert node di awal list
    void insertAwal(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = head;
        head = newNode;
    }
    // Insert node di akhir list
    void insertAkhir(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            return;
        }

        Node* temp = head;
        while (temp->next != NULL) {
```

```

        temp = temp->next;
    }
    temp->next = newNode;
}
// Insert node di tengah list
void insertTengah(string nama, int usia, int pos) {
    Node* newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;

    Node* temp = head;
    for (int i = 1; i < pos - 1; i++) {
        temp = temp->next;
    }

    newNode->next = temp->next;
    temp->next = newNode;
}
// Hapus node dengan nama Akechi
void hapusAkechi() {
    Node* temp = head;
    Node* prev = NULL;

    while (temp != NULL && temp->nama != "Akechi") {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        cout << "Data Akechi tidak ditemukan" << endl;
        return;
    }

    if (prev == NULL) {
        head = temp->next;
    } else {
        prev->next = temp->next;
    }

    delete temp;
}
// Tambahkan node antara John dan Jane
void tambahFutaba() {
    Node* temp = head;

```

```

        while (temp != NULL && temp->nama != "John") {
            temp = temp->next;
        }

        if (temp == NULL) {
            cout << "Data John tidak ditemukan" << endl;
            return;
        }

        Node* newNode = new Node;
        newNode->nama = "Futaba";
        newNode->usia = 18;
        newNode->next = temp->next;
        temp->next = newNode;
    }
    // Tambahkan node di awal list
    void tambahIgor() {
        Node* newNode = new Node;
        newNode->nama = "Igor";
        newNode->usia = 20;
        newNode->next = head;
        head = newNode;
    }
    // Ubah data Michael menjadi Reyn
    void ubahMichael() {
        Node* temp = head;

        while (temp != NULL && temp->nama != "Michael") {
            temp = temp->next;
        }

        if (temp == NULL) {
            cout << "Data Michael tidak ditemukan" << endl;
            return;
        }

        temp->nama = "Reyn";
        temp->usia = 18;
    }

    // Tampilkan seluruh data
    void tampilData() {
        Node* temp = head;

        if (temp == NULL) {

```

```

        cout << "List kosong" << endl;
        return;
    }

    while (temp != NULL) {
        cout << temp->nama << " " << temp->usia << endl;
        temp = temp->next;
    }
}

};

int main() {
    SingleLinkedList list;

    // Menambahkan data awal
    list.insertAkhir("John", 19);
    list.insertAkhir("Jane", 20);
    list.insertAkhir("Michael", 18);
    list.insertAkhir("Yusuke", 19);
    list.insertAkhir("Akechi", 20);
    list.insertAkhir("Hoshino", 18);
    list.insertAkhir("Karin", 18);

    // Hapus data Akechi
    list.hapusAkechi();

    // Tambahkan data Futaba antara John dan Jane
    list.tambahFutaba();

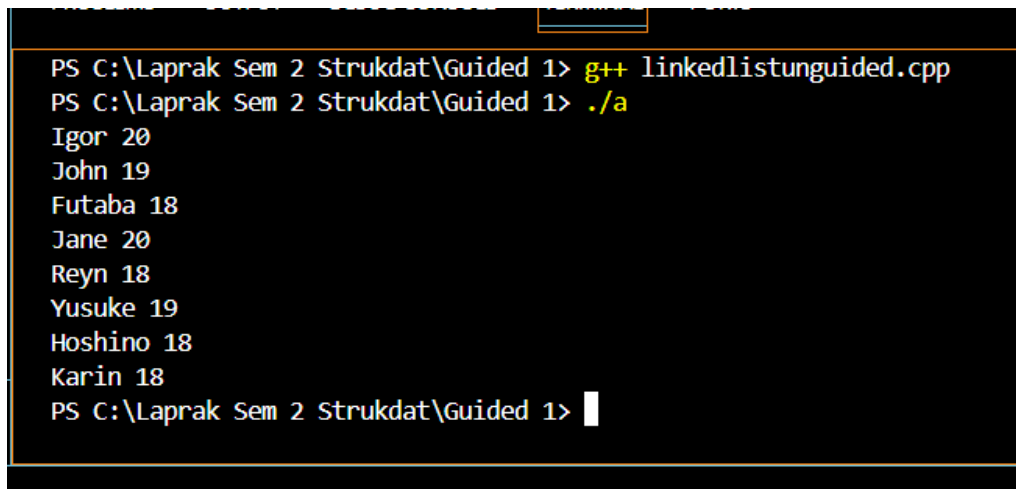
    // Tambahkan data Igor di awal list
    list.tambahIgor();

    // Ubah data Michael menjadi Reyn
    list.ubahMichael();

    // Tampilkan seluruh data
    list.tampilData();
    return 0;
}

```

Screenshot Program



```
PS C:\Laparak Sem 2 Strukdat\Guided 1> g++ linkedlistunguided.cpp
PS C:\Laparak Sem 2 Strukdat\Guided 1> ./a
Igor 20
John 19
Futaba 18
Jane 20
Reyn 18
Yusuke 19
Hoshino 18
Karin 18
PS C:\Laparak Sem 2 Strukdat\Guided 1> |
```

Deskripsi Program

Program diatas merupakan implementasi dari single linked list dimana program tersebut memiliki 3 elemen yaitu head, yang merupakan pointer untuk menunjuk ke node pertama dalam linked list. Dalam program juga terdapat insert awal, insert akhir dan insert tengah sebagai metode untuk memasukkan node baru kedalam linked list pada posisi awal, akhir atau tengah, 'hapusAkechi', 'tambahFutaba', 'tambahIgor', dan 'ubahMichael' sebagai metode untuk menghapus atau mengubah node pada linked list, serta 'tampilData' sebagai metode untuk menampilkan seluruh data yang terdapat pada linked list tersebut. Fungsi *main()* memiliki objek daftar dari kelas SingleLinkedList. Kemudian ada proses memasukkan data awal ke dalam linked list dengan menggunakan metode final *insert()*. Selanjutnya, metode *ubahAkechi()* dipanggil untuk menghapus node bernama "Akechi" dari linked list, dan kemudian metode *tambahFutaba()* dipanggil untuk menambahkan node baru "Futaba" di antaranya. Node "John" dan "Jane", panggil metode *tambahIgor()* untuk menambahkan node baru bernama "Igor" ke bagian atas daftar tertaut, dan panggil metode *ubahMichael()* untuk menambahkan node untuk mengubah nama "Michael" . "Untuk Reyn". Terakhir, metode *tampilData()* dipanggil, yang menampilkan semua data yang terdapat dalam linked list.

2. Unguided 2

Source Code

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

// Struktur untuk node produk
struct Product {
    string nama_produk;
    int harga;
    Product* prev;
    Product* next;
};

// Kelas untuk Double Linked List
class DoubleLinkedList {
private:
    Product* head;
    Product* tail;

public:
    DoubleLinkedList() {
        head = NULL;
        tail = NULL;
    }

    // Fungsi untuk menambahkan produk baru
    void addProduct(string nama_produk, int harga) {
        Product* newProduct = new Product;
        newProduct->nama_produk = nama_produk;
        newProduct->harga = harga;
        newProduct->prev = NULL;
        newProduct->next = NULL;

        if (head == NULL) {
            head = newProduct;
            tail = newProduct;
        } else {
            Product* current = head;
            while (current != NULL && current->nama_produk <
nama_produk) {
                current = current->next;
            }
        }
    }
};
```

```

        if (current == head) {
            newProduct->next = head;
            head->prev = newProduct;
            head = newProduct;
        } else if (current == NULL) {
            tail->next = newProduct;
            newProduct->prev = tail;
            tail = newProduct;
        } else {
            newProduct->next = current;
            newProduct->prev = current->prev;
            current->prev->next = newProduct;
            current->prev = newProduct;
        }
    }
}

// Fungsi untuk menghapus produk
void removeProduct(string nama_produk) {
    Product* current = head;
    while (current != NULL) {
        if (current->nama_produk == nama_produk) {
            if (current == head) {
                head = head->next;
                if (head != NULL) {
                    head->prev = NULL;
                }
            } else if (current == tail) {
                tail = tail->prev;
                tail->next = NULL;
            } else {
                current->prev->next = current->next;
                current->next->prev = current->prev;
            }
            delete current;
            return;
        }
        current = current->next;
    }
    cout << "Produk tidak ditemukan." << endl;
}

// Fungsi untuk memperbarui harga produk
void updatePrice(string nama_produk, int harga_baru) {
    Product* current = head;

```

```

        while (current != NULL) {
            if (current->nama_produk == nama_produk) {
                current->harga = harga_baru;
                return;
            }
            current = current->next;
        }
        cout << "Produk tidak ditemukan." << endl;
    }

    // Fungsi untuk menampilkan semua produk
    void displayProducts() {
        cout << "Nama Produk\tHarga" << endl;
        cout << "=====" << endl;
        Product* current = head;
        while (current != NULL) {
            cout << setw(12) << left << current->nama_produk <<
"\t" << current->harga << endl;
            current = current->next;
        }
        cout << endl;
    }

    // Fungsi untuk menambahkan produk di antara dua produk
    tertentu
    void addProductBetween(string prevProductName, string
newProductName, int newPrice) {
        Product* current = head;
        while (current != NULL) {
            if (current->nama_produk == prevProductName) {
                Product* newProduct = new Product;
                newProduct->nama_produk = newProductName;
                newProduct->harga = newPrice;
                newProduct->prev = current;
                newProduct->next = current->next;
                if (current->next != NULL) {
                    current->next->prev = newProduct;
                }
                current->next = newProduct;
                return;
            }
            current = current->next;
        }
        cout << "Produk sebelumnya tidak ditemukan." << endl;
    }
}

```



```

};

int main() {
    DoubleLinkedList productList;

    // Menambahkan produk ke dalam linked list
    productList.addProduct("Originote", 60000);
    productList.addProduct("Somethinc", 150000);
    productList.addProduct("Skintific", 100000);
    productList.addProduct("Wardah", 50000);
    productList.addProduct("Hanasui", 30000);

    int choice;
    string prevProductName, newProductName;
    int newPrice;

    do {
        cout << "\nToko Skincare Purwokerto" << endl;
        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 << "Pilih menu: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Masukkan Nama Produk: ";
                cin >> newProductName;
                cout << "Masukkan Harga: ";
                cin >> newPrice;
                productList.addProduct(newProductName,
newPrice);
                break;
            case 2:
                cout << "Masukkan Nama Produk yang Ingin
Dihapus: ";
                cin >> newProductName;
                productList.removeProduct(newProductName);
                break;
            case 3:

```

```

        cout << "Masukkan Nama Produk yang Ingin
Diupdate: ";
        cin >> newProductName;
        cout << "Masukkan Harga Baru: ";
        cin >> newPrice;
        productList.updatePrice(newProductName,
newPrice);
        break;
    case 4:
        cout << "Masukkan Nama Produk Sebelumnya: ";
        cin >> prevProductName;
        cout << "Masukkan Nama Produk Baru: ";
        cin >> newProductName;
        cout << "Masukkan Harga Baru: ";
        cin >> newPrice;
        productList.addProductBetween(prevProductName,
newProductName, newPrice);
        break;
    case 5:
        // Implementasi kode untuk menghapus data pada
posisi tertentu jika diperlukan
        break;
    case 6:
        // Implementasi kode untuk menghapus semua data
jika diperlukan
        break;
    case 7:
        cout << "Data Produk:" << endl;
        productList.displayProducts();
        break;
    case 8:
        cout << "Terima kasih!" << endl;
        break;
    default:
        cout << "Pilihan tidak valid. Silakan pilih
lagi." << endl;
    }
} while (choice != 8);

return 0;
}

```

Screenshot Program

```
-----
Nama Produk  **  Harga
-----
Originote    60000
Somethinc    150000
Azarine 65000
Skintific    100000
Hanasui 30000
-----

* 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
Pilih: 3
Nama Produk yang akan diupdate: Hanasui
Nama Baru: Cleora
Harga Baru: 55000
-----

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

```
-----

* 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
Pilih: 3
Nama Produk yang akan diupdate: Hanasui
Nama Baru: Cleora
Harga Baru: 55000
-----

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

* 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
```

```
-----
* 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
Pilih: 7
-----
Nama Produk  **  Harga
-----
Originote      60000
Somethinc      150000
Azarine 65000
Skintific      100000
Cleora  55000
-----
* 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
Pilih: █
```

Deskripsi Program

Program diatas mengimplementasikan struktur data linked list ganda. Program ini memungkinkan pengguna untuk melakukan berbagai operasi pada linked list seperti menambahkan node, menyisipkan node setelah node tertentu, menghapus node, memperbarui node, menampilkan daftar, dan menghapus seluruh daftar. Program ini mendefinisikan dua kelas, Node dan DoublyLinkedList. Kelas Node mewakili sebuah node dalam linked list dan memiliki tiga variabel anggota: nama_produk (sebuah string yang mewakili nama produk), harga (sebuah bilangan bulat yang mewakili harga produk), prev (sebuah pointer ke node sebelumnya dalam linked list), dan next (sebuah pointer ke node berikutnya dalam linked list). Kelas DoublyLinkedList mewakili linked list ganda dan memiliki dua variabel anggota: head (sebuah pointer ke node pertama dalam linked list) dan tail (sebuah pointer ke node terakhir dalam linked list). Kelas ini juga mendefinisikan beberapa fungsi anggota yang memungkinkan pengguna untuk memanipulasi daftar. Fungsi push menambahkan node baru ke awal daftar. Fungsi insertAfter menyisipkan node baru setelah node tertentu dalam daftar. Fungsi pop menghapus node tertentu dari daftar. Fungsi update memperbarui nilai nama_produk dan harga dari sebuah node dengan

nilai baru. Fungsi `deleteAll` menghapus seluruh daftar dengan menghapus semua node. Fungsi `display` mencetak nilai `nama_produk` dan harga dari seluruh node dalam daftar. Fungsi utama dari program ini berisi loop yang menampilkan menu pilihan kepada pengguna dan memungkinkan pengguna untuk melakukan berbagai operasi pada linked list berdasarkan pilihan mereka. Pengguna dapat memilih untuk menambahkan node, menyisipkan node setelah node tertentu, menghapus node, memperbarui node, menampilkan daftar, atau menghapus seluruh daftar. Loop akan berlanjut hingga pengguna memilih untuk keluar dari program.

KESIMPULAN

Single Linked List dan Doble Linked List adalah dua struktur data yang digunakan untuk menyimpan data dalam bentuk daftar tertaut. Pada single linked list setiap node hanya memiliki satu pointer node berikutnya, sedangkan pada double linked list setiap node memiliki dua pointer, yaitu pointer node berikutnya dan pointer node sebelumnya.

Keuntungan dari Single Linked List adalah struktur data ini lebih sederhana dan lebih efisien dalam hal penggunaan memori karena setiap node hanya menyimpan satu pointer. Namun, kelemahan dari Single Linked List adalah sulitnya mengakses node sebelumnya.

Pada saat yang sama, double linked list memiliki keuntungan akses mudah ke node sebelumnya, membuatnya lebih mudah untuk melakukan operasi seperti menghapus node pada posisi tertentu. Namun, kelemahan dari double linked list adalah bahwa struktur data ini membutuhkan lebih banyak memori karena setiap node menyimpan dua pointer.

Keduanya digunakan untuk memudahkan penyimpanan data, terutama dalam kasus di mana jumlah datanya besar, dan untuk memudahkan pemrosesan data.