

LAPORAN PRAKTIKUM
MODUL 3
SINGLE AND DOUBLE LINKED LIST



Disusun oleh:
Rasyid Nafsyarie
NIM : 2311102011

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

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
2023

BAB I

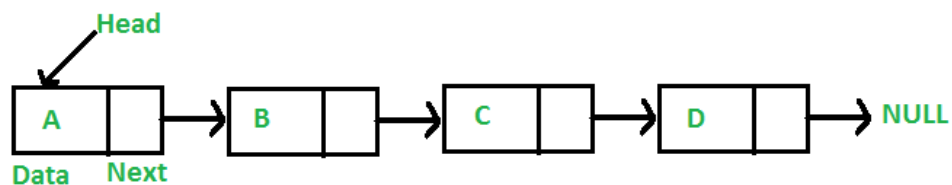
TUJUAN PRAKTIKUM

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

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.

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->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;
}

```

SCREENSHOOT PROGRAM

3	satu						
3	satu	5	dua				
2	tiga	3	satu	5	dua		
1	empat	2	tiga	3	satu	5	dua
2	tiga	3	satu	5	dua		
2	tiga	3	satu				
2	tiga	7	lima	3	satu		
2	tiga	3	satu				
1	enam	3	satu				
1	enam	8	tujuh				
1	enam	11	tujuh				

DESKRIPSI PROGRAM

Dalam program ini, kita telah berhasil mengimplementasikan Linked List tunggal non-circular menggunakan bahasa pemrograman C++. Kita dapat menambahkan, menghapus, mengubah, dan menampilkan simpul dalam Linked List menggunakan fungsi-fungsi yang telah kita buat. Linked List adalah struktur data yang fleksibel dan efisien untuk mengelola kumpulan data yang dinamis. Dengan pemahaman yang baik tentang Linked List, kita dapat mengembangkan program yang lebih kompleks dan efisien.

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;
}

```

SCREENSHOOT PROGRAM

<pre> 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: 1 Enter kata to add: budi </pre>	<p>DESKRIPSI PROGRAM</p> <p>Dalam program C++ yang dibuat, program ini memiliki array dengan kapasitas maksimum dalam sebuah array yang akan meminta pengguna untuk memasukkan data, kemudian akan menampilkan nilai data yang dimasukkan.</p>
--	---

2.1. Ketika program menambahkan kata

<pre> 1. Add data 2. Delete data 3. Update data 4. Clear data 5. Display data 6. Exit Enter your choice: 5 2 budi </pre>	<p>Enter your choice: 5 Enter data to add: 2 Enter kata to add: budi</p> <p> 2.1. Ketika program menambahkan kata</p> <p>DESKRIPSI PROGRAM</p> <p>Dalam program C++ yang dibuat, program ini memiliki array dengan kapasitas maksimum dalam sebuah array yang akan meminta pengguna untuk memasukkan data, kemudian akan menampilkan nilai data yang dimasukkan.</p>
--	--

2.2. Ketika Program menampilkan data

DESKRIPSI PROGRAM

Pertama, kita mendefinisikan class Node yang merepresentasikan simpul dalam Doubly Linked List. Setiap simpul memiliki atribut data dan kata untuk menyimpan data dan kata, serta pointer prev dan next untuk menghubungkan simpul dengan simpul sebelumnya dan setelahnya. Selanjutnya, kita mendefinisikan class DoublyLinkedList yang merepresentasikan Doubly Linked List itu sendiri. Class ini memiliki atribut head dan tail yang merupakan pointer ke simpul pertama dan terakhir dalam Doubly Linked List.

UNGUIDED

1. UNGUIDED 1

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

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

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

class LinkedList {
private:
    Node* head;

public:
```

```
LinkedList() {
    head = nullptr;
}

void insertAwal(string nama, int usia) {
    Node* newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;
    newNode->next = head;
    head = newNode;
}

void insertAkhir(string nama, int usia) {
    Node* newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;
    newNode->next = nullptr;

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

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertSetelah(string nama, int usia, string
namaSebelum) {
    Node* newNode = new Node;
    newNode->nama = nama;
```

```

newNode->usia = usia;

Node* temp = head;
while (temp != nullptr && temp->nama != namaSebelum) {
    temp = temp->next;
}

if (temp == nullptr) {
    cout << "Node dengan nama " << namaSebelum << " tidak
ditemukan." << endl;
    return;
}

newNode->next = temp->next;
temp->next = newNode;
}

void hapus(string nama) {
    if (head == nullptr) {
        cout << "Linked list kosong." << endl;
        return;
    }

    if (head->nama == nama) {
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    Node* prev = head;
    Node* temp = head->next;
    while (temp != nullptr && temp->nama != nama) {
        prev = temp;
    }
}

```

```

        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
        return;
    }

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

void ubah(string nama, string namaBaru, int usiaBaru) {
    Node* temp = head;
    while (temp != nullptr && temp->nama != nama) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
        return;
    }

    temp->nama = namaBaru;
    temp->usia = usiaBaru;
}

void tampilkan() {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->nama << " " << temp->usia << endl;
        temp = temp->next;
    }
}

```

```

    }

    }

};

int main() {
    LinkedList myList;

    myList.insertAwal("Rasyid Nafsyarie", 19);
    myList.insertAwal("John", 19);
    myList.insertAwal("Jane", 20);
    myList.insertAwal("Michael", 18);
    myList.insertAwal("Yusuke", 19);
    myList.insertAwal("Akechi", 20);
    myList.insertAwal("Hoshino", 18);
    myList.insertAwal("Karin", 18);

    cout << "Data setelah langkah (a):" << endl;
    myList.tampilkan();
    cout << endl;

    myList.hapus("Akechi");
    myList.insertSetelah("Futaba", 18, "John");
    myList.insertAwal("Igor", 20);
    myList.ubah("Michael", "Reyn", 18);
    cout << "Data setelah dilakukan semua operasi:" << endl;
    myList.tampilkan();

    return 0;
}

```


SCREENSHOOT PROGRAM



```
Karin 18
Hoshino 18
Akechi 20
Yusuke 19
Michael 18
Jane 20
John 19
Rasyid Hafsyarie 19

Data setelah dilakukan semua operasi:
Karin 18
Hoshino 18
Yusuke 19
Reyn 18
Jane 20
John 19
Futaba 18
Rasyid Hafsyarie 19

Data Array: 1 2 3 4 5 6 7 8 9 10
Input Array: 2 4 6 8 10
Input Array: 1 1 1 1 1
```

DESKRIPSI PROGRAM

Dalam contoh kode di atas, kita telah melihat implementasi Linked List dalam bahasa pemrograman C++. Kita dapat memasukkan node baru di awal, akhir, atau setelah simpul tertentu dalam Linked List. Kita juga dapat menghapus node berdasarkan nama dan mengubah data (nama dan usia) dari node tertentu. Selain itu, kita dapat menampilkan semua data dalam Linked List. Linked List adalah struktur data yang fleksibel dan efisien untuk menyimpan dan mengelola kumpulan data yang dinamis.

2. UNGUIDED 2

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
Originote	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>
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 pop() {
    if (head == nullptr) {
        return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
        head->prev = nullptr;
    } else {
        tail = nullptr;
    }
    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;
}

void insertAfter(string namaProduk, int harga, string
afterNamaProduk) {
    Node* newNode = new Node;
    newNode->namaProduk = namaProduk;
    newNode->harga = harga;

```

```

Node* current = head;
while (current != nullptr) {
    if (current->namaProduk == afterNamaProduk) {
        newNode->prev = current;
        newNode->next = current->next;
        if (current->next != nullptr) {
            current->next->prev = newNode;
        } else {
            tail = newNode;
        }
        current->next = newNode;
        return;
    }
    current = current->next;
}

void deleteAfter(string afterNamaProduk) {
    Node* current = head;
    while (current != nullptr) {
        if (current->namaProduk == afterNamaProduk) {
            Node* temp = current->next;
            if (temp != nullptr) {
                current->next = temp->next;
                if (temp->next != nullptr) {
                    temp->next->prev = current;
                } else {
                    tail = current;
                }
                delete temp;
            }
            return;
        }
        current = current->next;
    }
}

```

```

    }

}

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;
    cout << "Nama Produk\tHarga" << endl;
    while (current != nullptr) {
        cout << current->namaProduk << "\t\t" << current->
>harga << endl;
        current = current->next;
    }
    cout << endl;
}

};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "Toko 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;
int choice;
cout << "Masukkan pilihan Anda: ";
cin >> choice;
switch (choice) {
    case 1: {
        string namaProduk;
        int harga;
        cout << "Masukkan nama produk: ";
        cin >> namaProduk;
        cout << "Masukkan harga: ";
        cin >> harga;
        list.push(namaProduk, harga);
        break;
    }
    case 2: {
        list.pop();
        break;
    }
    case 3: {
        string oldNamaProduk, newNamaProduk;
        int newHarga;
        cout << "Masukkan nama produk yang lama: ";
        cin >> oldNamaProduk;
        cout << "Masukkan nama produk yang baru: ";
        cin >> newNamaProduk;
        cout << "Masukkan harga yang baru: ";
        cin >> newHarga;
        bool updated = list.update(oldNamaProduk,
newNamaProduk, newHarga);
        if (!updated) {
```

```

        cout << "Data tidak ditemukan" << endl;
    }
    break;
}
case 4: {
    string namaProduk, afterNamaProduk;
    int harga;
    cout << "Masukkan nama produk: ";
    cin >> namaProduk;
    cout << "Masukkan harga: ";
    cin >> harga;
    cout << "Masukkan nama produk setelahnya: ";
    cin >> afterNamaProduk;
    list.insertAfter(namaProduk, harga,
afterNamaProduk);
    break;
}
case 5: {
    string afterNamaProduk;
    cout << "Masukkan nama produk setelahnya: ";
    cin >> afterNamaProduk;
    list.deleteAfter(afterNamaProduk);
    break;
}
case 6: {
    list.deleteAll();
    break;
}
case 7: {
    list.display();
    break;
}
case 8: {
    return 0;
}

```

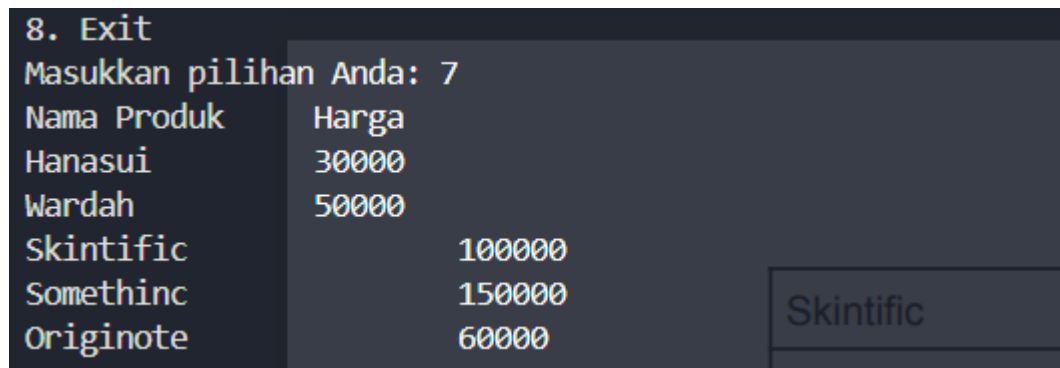


```

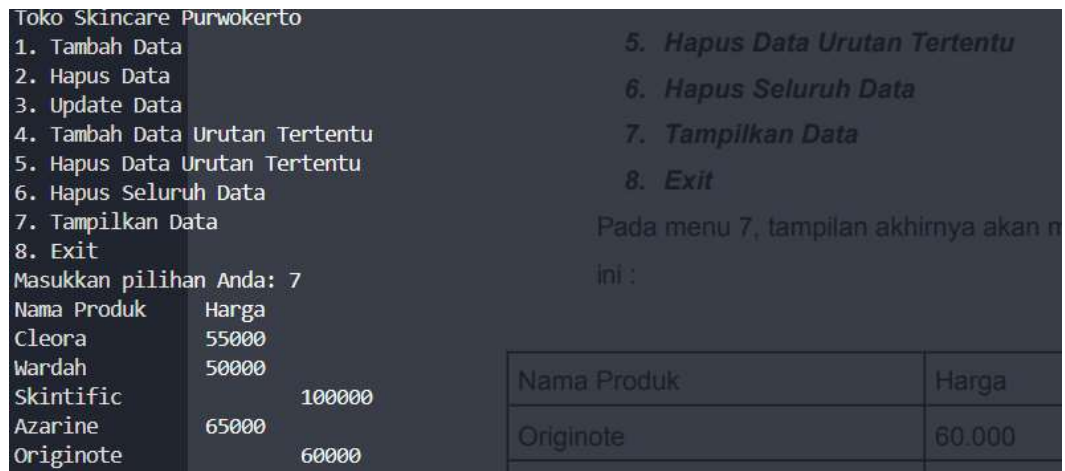
    }
    default: {
        cout << "Pilihan tidak valid" << endl;
        break;
    }
}
}
return 0;
}

```

SCREENSHOOT PROGRAM



2.1. Tampilan sebelum data diupdate



2.2. Tampilan setelah update data

DESKRIPSI PROGRAM

Dalam program ini, kita telah membuat sebuah program untuk mengelola data produk skincare di sebuah toko. Program ini menggunakan struktur data Doubly Linked List untuk menyimpan dan mengatur data produk. Program ini memungkinkan pengguna untuk menambah, menghapus, mengupdate, dan menampilkan data produk skincare. Dengan menggunakan struktur data Doubly Linked List, kita dapat dengan mudah melakukan operasi pada data produk dengan efisien.

BAB IV

KESIMPULAN

Setelah melakukan pembelajaran mengenai tipe data di Bahasa Pemrograman C++ berikut poin utama yang telah dipelajari :

1. Single Linked List : Struktur data berupa kumpulan node yang terhubung secara sekuensial dengan menggunakan pointer.
2. Single Linked List : Setiap node memiliki dua bagian: bagian isi (data) dan bagian pointer yang menunjuk ke node berikutnya.
3. Double Linked List : Representasi struktur data Double Linked List mencakup pointer HEAD yang menunjuk pada node pertama dan pointer TAIL yang menunjuk pada node terakhir.
4. Double Linked List: Linked list dikatakan kosong jika nilai pointer head adalah NULL, dan nilai pointer prev dari HEAD serta nilai pointer next dari TAIL selalu NULL.

DAFTAR PUSTAKA

Trivusi. (2022, 16 September). Struktur Data Linked List: Pengertian, Karakteristik, dan Jenis-jenisnya, Diakses pada 29 Maret 2024 dari <https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html>