

**LAPORAN PRAKTIKUM STRUKTUR DATA DAN
ALGORITMA
MODUL 3
SINGLE AND DOUBLE LINKED LIST**



DISUSUN OLEH:

PRIESTY AMEILIANA MAULIDAH

2311102175

S1 IF-11-E

DOSEN:

Muhammad Afrizal Amrustian, S. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO PURWOKERTO

2024

A.DASAR TEORI

a.)Single Linked List

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

b.)Double Linked List

Double Linked List adalah struktur data Linked List yang mirip dengan Single Linked List, namun dengan tambahan satu pointer tambahan pada setiap simpul

yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul mana saja secara efisien. Setiap simpul pada Double Linked List memiliki tiga elemen penting, yaitu elemen data , pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya.

B.Guided

Guided 1

```
// NAMA : PRIESTY AMEILIANA MAULIDAH

// NIM   : 2311102175

#include <iostream>

using namespace std;

///PROGRAM SINGLE LINKED LIST NON-CIRCULAR //Deklarasi Struct Node

struct Node{

//komponen/member

int data;

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

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

```
//Hitung Jumlah List

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

//Tambah Tengah

void insertTengah(int data, 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;

// tranversing
bantu = head;

int nomor = 1;

while( nomor < posisi - 1 ){
baru = 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{
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){
if (isEmpty() == false){
head->data = data;
}
else{
cout << "List masih kosong!" << endl;
}
}
```

```
//Ubah Tengah
void ubahTengah(int data, 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;
        }
    }
    else{
        cout << "List masih kosong!" << endl;
    }
}
```

```
//Ubah Belakang
void ubahBelakang(int data){
    if (isEmpty() == false){
        tail->data = data;
    }
    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 << ends;

    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();

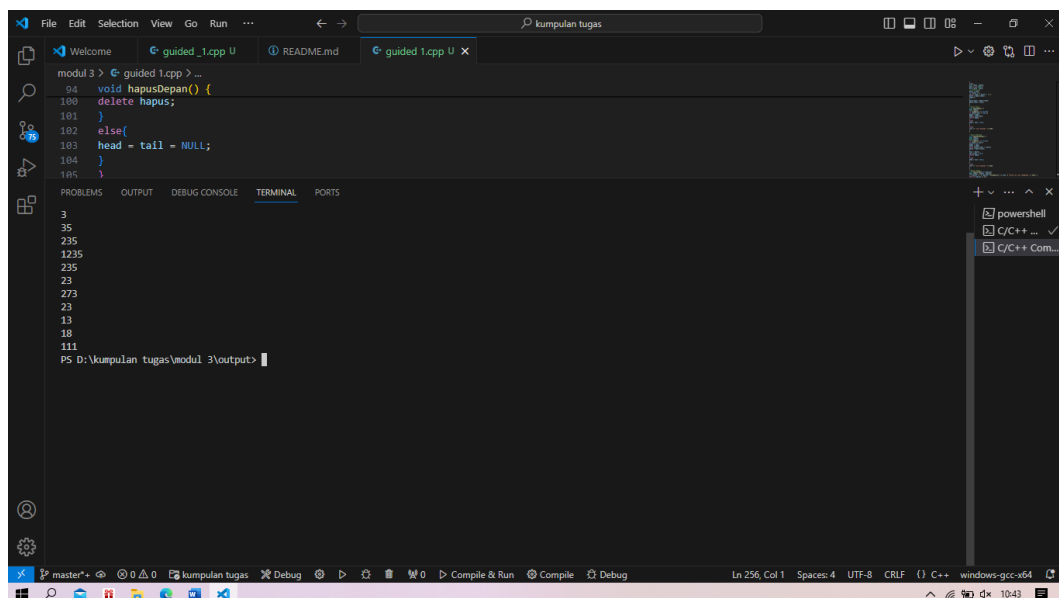
ubahTengah(11, 2);

tampil();

return 0;

}
```

Screenshots output



Deskripsi:

Operasi Menambahkan Data:

- insertDepan(int nilai): Menambahkan data baru ke depan list.
- insertBelakang(int nilai): Menambahkan data baru ke belakang list.

- insertTengah(int data, int posisi): Menambahkan data baru di posisi tertentu dalam list.

Operasi Menghapus Data:

- hapusDepan(): Menghapus data dari depan list.
- hapusBelakang(): Menghapus data dari belakang list.
- hapusTengah(int posisi): Menghapus data di posisi tertentu dalam list.

Operasi Mengubah Data:

- ubahDepan(int data): Mengubah data di depan list.
- ubahTengah(int data, int posisi): Mengubah data di posisi tertentu dalam list.
- ubahBelakang(int data): Mengubah data di belakang list.

Operasi Lainnya:

- isEmpty(): Mengecek apakah list kosong.
- hitungList(): Menghitung jumlah data dalam list.
- tampil(): Menampilkan seluruh data dalam list.
- clearList(): Menghapus seluruh data dalam list.

Guided 2

```
// NAMA : PRIESTY AMEILIANA MAULIDAH
// NIM : 2311102175
#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 = 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

```

D:\kumpulan tugas\guided satu\bin\Debug\guided satu.exe
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter data to add: 5
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: 3
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 5
Enter new data: 15
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:

```

Deskripsi :

1. **Baris** #include <iostream> **dan** using namespace std;
 - Baris pertama menyertakan berkas header <iostream> yang menyediakan fungsi untuk input/output seperti cin (untuk

membaca input) dan cout (untuk menampilkan output) pada konsol.

- Baris kedua menggunakan namespace std. Ini memungkinkan Anda untuk menggunakan objek standar C++ seperti cin, cout, endl, dll., tanpa perlu awalan std::.

2. **Kelas Node**

- Kelas ini merepresentasikan sebuah node tunggal dalam Linked List Ganda.
 - data: Variabel anggota bertipe integer untuk menyimpan nilai data dari node tersebut.
 - prev: Pointer ke node sebelumnya di dalam list.
 - next: Pointer ke node selanjutnya di dalam list.

3. **Kelas DoublyLinkedList**

- Kelas ini mengelola struktur data Linked List Ganda.
 - head: Pointer ke node pertama (head) di dalam list.

- tail: Pointer ke node terakhir (tail) di dalam list.
- Fungsi anggota publik:
 - DoublyLinkedList(): Konstruktor yang menginisialisasi Linked List kosong dengan menyetel head dan tail ke nullptr.
 - push(int data): Fungsi ini menambahkan node baru yang berisi data yang diberikan di awal (head) list. Fungsi ini menangani kasus untuk list kosong dan list tidak kosong, memperbarui pointer sesuai kebutuhan.
 - pop(): Fungsi ini menghapus node pertama (head) dari list. Fungsi ini memeriksa apakah list kosong dan menangani pembaruan pointer tail jika list menjadi kosong setelah penghapusan.
 - update(int oldData, int newData): Fungsi ini mencari node yang berisi oldData dan memperbarui datanya dengan newData. Fungsi ini mengembalikan true jika

pembaruan

berhasil, false sebaliknya (data tidak ditemukan).

- `deleteAll()`: Fungsi ini menghapus semua node di dalam list, iterasi melalui list dan membebaskan memori untuk setiap node. Fungsi ini menyetel head dan tail ke `nullptr` untuk mencerminkan list kosong.
- `display()`: Fungsi ini mencetak data dari semua node di dalam list, dimulai dari head dan iterasi hingga pointer null ditemui.

4. **Fungsi** main

- Ini adalah titik masuk program.
- Sebuah instance dari kelas `DoublyLinkedList` bernama `list` dibuat.
- Loop tak hingga (`while (true)`) digunakan untuk menampilkan menu kepada pengguna dengan enam pilihan:

1. Tambah data
2. Hapus data
3. Perbarui data

4. Bersihkan data
 5. Tampilkan data
 6. Keluar
- Pilihan pengguna disimpan dalam variabel choice.
 - Pernyataan switch digunakan untuk menangani pilihan pengguna:
 - Kasus 1 (choice == 1): Pengguna memasukkan data untuk ditambahkan. Fungsi push dipanggil untuk menambahkan node baru.
 - Kasus 2 (choice == 2): Fungsi pop dipanggil untuk menghapus node pertama.
 - Kasus 3 (choice == 3): Pengguna memasukkan nilai data lama dan baru. Fungsi update dipanggil untuk mencari dan memperbarui data.
 - Kasus 4 (choice == 4): Fungsi deleteAll dipanggil untuk menghapus semua data di dalam list.
 - Kasus 5 (choice == 5): Fungsi display dipanggil untuk mencetak isi list.

- Kasus 6 (choice == 6): Loop keluar, mengakhiri program (return 0;).
- Default: Pesan kesalahan ditampilkan untuk pilihan yang tidak valid.

c. unguided/tugas

unguided 1

```
// nama : priesty ameiliana mulidah
// nim : 2311102175

#include <iostream>
#include <string>

using namespace std;

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

    Node(string nama, int usia) {
        this->nama = nama;
        this->usia = usia;
        next = nullptr;
    }
};

class SingleLinkedList {
private:
    Node* head;
    Node* tail;
```

```

public:

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

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

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

void insertTengah(string nama, int usia, string namaSebelum) {
    Node* newNode = new Node(nama, usia);
    Node* current = head;

    while (current != nullptr && current->nama != namaSebelum) {
        current = current->next;
    }
}

```

```

if (current == nullptr) {
    cout << "Data " << namaSebelum << " tidak ditemukan!" << endl;
} else {
    newNode->next = current->next;
    current->next = newNode;
}
}

```

```

void hapusData(string nama) {
    Node* current = head;
    Node* prev = nullptr;
    if (current == nullptr) {
        cout << "Data " << nama << " tidak ditemukan!" << endl;
    } else if (current == head) {
        head = head->next;
        if (head == nullptr) {
            tail = nullptr;
        }
    } else if (current == tail) {
        prev->next = nullptr;
        tail = prev;
    } else {
        prev->next = current->next;
    }

    delete current;
}

```

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

```
bool isEmpty() {  
    return head == nullptr;  
}  
};
```

```
int main() {  
    SingleLinkedList list;  
  
    // Masukkan data diri  
    string namaAnda;  
    int usiaAnda;  
    cout << "Masukkan nama Anda: ";  
    cin >> namaAnda;  
    cout << "Masukkan usia Anda: ";  
    cin >> usiaAnda;  
    list.insertDepan(namaAnda, usiaAnda);
```

```
// Masukkan data lainnya

string nama[] = {"John", "Jane", "Michael", "Yusuke", "Hoshino", "Karin"};
int usia[] = {19, 20, 18, 19, 18, 18};
for (int i = 0; i < 6; i++) {
    list.insertBelakang(nama[i], usia[i]);
}

// b. Hapus data Akechi
list.hapusData("Akechi");

// c. Tambahkan data Futaba diantara John dan Jane
list.insertTengah("Futaba", 18, "John");

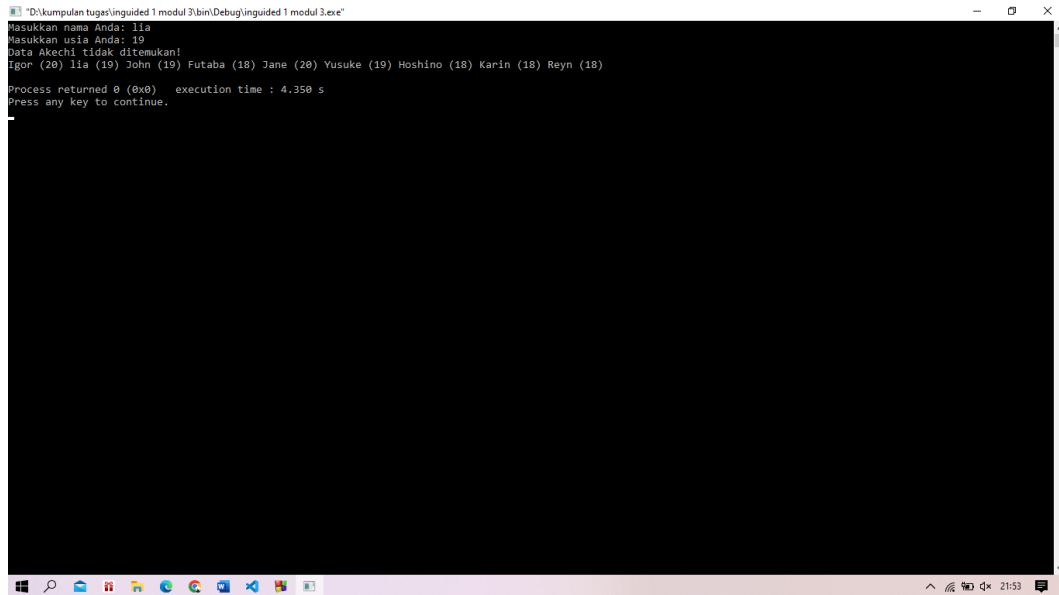
// d. Tambahkan data Igor diawal
list.insertDepan("Igor", 20);

// e. Ubah data Michael menjadi Reyn
list.hapusData("Michael");
list.insertBelakang("Reyn", 18);

// f. Tampilkan seluruh data
list.tampilData();

return 0;
}
```

Screenshots output:



```
D:\kumpulan tugas\inguided 1 modul 3\bin\Debug\inguided 1 modul 3.exe
Masukkan nama Anda: Iia
Masukkan usia Anda: 19
Data Akechi tidak ditemukan!
Egor (20) Iia (19) John (19) Futaba (18) Jane (20) Yusuke (19) Hoshino (18) Karin (18) Reyn (18)
Process returned 0 (0x0) execution time : 4.350 s
Press any key to continue.
```

Deskripsi :

Deskripsi Program Single Linked List Non-Circular

Komponen Program:

- **Baris #include <iostream> dan #include <string>:**
 - Baris pertama menyertakan berkas header <iostream> yang menyediakan fungsi untuk input/output seperti cin (untuk membaca input) dan cout (untuk menampilkan output) pada konsol.
 - Baris kedua menyertakan berkas header <string> yang menyediakan class string untuk penanganan string (teks).
- **struct Node:**

- Merupakan struktur data yang digunakan untuk membangun blok penyusun Single Linked List. Setiap node memiliki:
 - nama: Variabel bertipe string untuk menyimpan nama mahasiswa.
 - usia: Variabel bertipe int untuk menyimpan usia mahasiswa.
 - next: Pointer yang menunjuk ke node berikutnya di dalam Linked List.
- Konstruktor Node(string nama, int usia) menginisialisasi nilai nama dan usia saat pembuatan node baru.
- class SingleLinkedList:
 - Mengelola keseluruhan Single Linked List Non-Circular.
 - head: Pointer yang menunjuk ke node pertama (posisi kepala) di dalam Linked List.
 - tail: Pointer yang menunjuk ke node terakhir (posisi ekor) di dalam Linked List.
 - Fungsi anggota publik:

- `SingleLinkedList()`: Konstruktor yang menginisialisasi Linked List kosong dengan menyetel head dan tail ke nullptr (menandakan tidak ada node).
- `insertDepan(string nama, int usia)`: Fungsi untuk menambahkan node baru yang berisi data nama dan usia di awal (posisi kepala) Linked List.
- `insertBelakang(string nama, int usia)`: Fungsi untuk menambahkan node baru yang berisi data nama dan usia di akhir (posisi ekor) Linked List.
- `insertTengah(string nama, int usia, string namaSebelum)`: Fungsi untuk menambahkan node baru yang berisi data nama dan usia diantara node dengan nama namaSebelum.
- `hapusData(string nama)`: Fungsi untuk menghapus node yang memiliki nama nama dari Linked List.
- `tampilData()`: Fungsi untuk menampilkan data (nama dan usia) dari semua node di dalam Linked List,

dimulai dari posisi kepala hingga akhir (posisi ekor).

- isEmpty(): Fungsi untuk mengecek apakah Linked List kosong (tidak ada node).

- **Fungsi main:**

- Fungsi utama tempat program dijalankan.
- Dibuat instance dari class SingleLinkedList bernama list untuk merepresentasikan Linked List yang akan digunakan untuk menyimpan data mahasiswa.
- Meminta input nama dan usia pengguna, lalu menambahkannya ke Linked List menggunakan insertDepan.
- Menambahkan data mahasiswa lain yang sudah disediakan ke Linked List menggunakan insertBelakang.
- Melakukan operasi sesuai dengan instruksi soal pada Linked List:
 - Menghapus data "Akechi".
 - Menambahkan data "Futaba" diantara "John" dan "Jane".

- Menambahkan data "Igor" di awal.
- Mengubah data "Michael" menjadi "Reyn".
- Menampilkan seluruh data mahasiswa yang tersimpan di Linked List menggunakan tampilData.

Unguided 2

```
// nama : priesty ameiiana mualidah
// nim : 2311102175

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

// Node class for the linked list
class Node {
public:
    string namaProduk;
    int harga;
    Node* prev;
    Node* next;

    // Constructor to initialize node
    Node(string namaProduk, int harga) {
        this->namaProduk = namaProduk;
        this->harga = harga;
        prev = nullptr;
        next = nullptr;
    }
};

// Linked list class
class DoubleLinkedList {
private:
    Node* head;
    Node* tail
```

```
public:

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

    // Function to insert data at the beginning of the list
    void insertDepan(string namaProduk, int harga) {
        Node* newNode = new Node(namaProduk, harga);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    // Function to insert data at the end of the list
    void insertBelakang(string namaProduk, int harga) {
        Node* newNode = new Node(namaProduk, harga);
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
}
```

```

// Function to insert data after a specific node

void insertTengah(string namaProduk, int harga, string namaProdukSebelum) {
    Node* newNode = new Node(namaProduk, harga);
    Node* current = head;

    while (current != nullptr && current->namaProduk != namaProdukSebelum) {
        current = current->next;
    }

    if (current == nullptr) {
        cout << "Data " << namaProdukSebelum << " tidak ditemukan!" << endl;
    } else {
        newNode->next = current->next;
        if (current->next != nullptr) {
            current->next->prev = newNode;
        } else {
            tail = newNode;
        }
        current->next = newNode;
        newNode->prev = current;
    }
}

// Function to delete a node with specific name

void hapusData(string namaProduk) {
    Node* current = head;
    Node* prev = nullptr;

```

```
while (current != nullptr && current->namaProduk != namaProduk) {  
    prev = current;  
    current = current->next;  
}  
  
if (current == nullptr) {  
    cout << "Data " << namaProduk << " tidak ditemukan!" << endl;  
} else if (current == head) {  
    head = head->next;  
    if (head == nullptr) {  
        tail = nullptr;  
    } else {  
        head->prev = nullptr;  
    }  
} else if (current == tail) {  
    prev->next = nullptr;  
    tail = prev;  
} else {  
    prev->next = current->next;  
    current->next->prev = prev;  
}  
  
delete current;  
}
```

```
// Function to update data with specific name

void updateData(string namaProduk, string namaProdukBaru, int hargaBaru) {

    Node* current = head;

    while (current != nullptr && current->namaProduk != namaProduk) {

        current = current->next;

    }

    if (current == nullptr) {

        cout << "Data " << namaProduk << " tidak ditemukan!" << endl;

    } else {

        current->namaProduk = namaProdukBaru;

        current->harga = hargaBaru;

    }

}

// Function to delete data at specific position

void hapusDataUrutan(int posisi) {

    if (isEmpty()) {

        cout << "List kosong!" << endl;

        return;

    }

}
```

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



```

delete current;

    }

}

// Function to delete all nodes in the list
void hapusSeluruhData() {
    Node* current = head;
    while (current != nullptr) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    head = nullptr;
    tail = nullptr;
}

// Function to display all data in the list
void tampilData() {
    cout << "Nama Produk Harga" << endl;
    Node* current = head;
    while (current != nullptr) {
        cout << current->namaProduk << " " << current->harga << endl;
        current = current->next;
    }
}

```

```

// Function to display all data in the list

void tampilData() {

    cout << "Nama Produk Harga" << endl;

    Node* current = head;

    while (current != nullptr) {

        cout << current->namaProduk << " " << current->harga <<
endl;

        current = current->next;

    }

}

// Function to check if the list is empty

bool isEmpty() {

    return head == nullptr;

}

};

int main() {

    DoubleLinkedList list;

    // Data awal

    list.insertDepan("Originote", 60000);

    list.insertBelakang("Somethinc", 150000);

    list.insertBelakang("Skintific", 100000);

    list.insertBelakang("Wardah", 50000);

    list.insertBelakang("Hanasui", 30000);

```

```
int choice;

do {

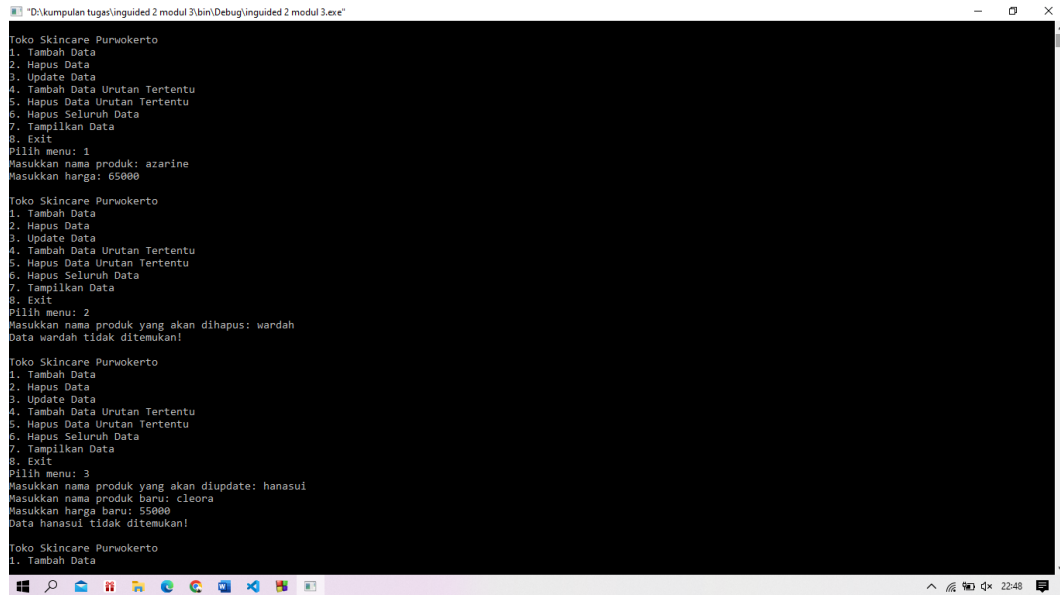
    cout << "\nToko Skincare Purwokerto" << endl;
    cout << "1. Tambah Data" << endl;
    cout << "2. Hapus Data" << endl;
    cout << "3. Update Data" << endl;
    cout << "4. Tambah Data Urutan Tertentu" << endl;
    cout << "5. Hapus Data Urutan Tertentu" << endl;
    cout << "6. Hapus Seluruh Data" << endl;
    cout << "7. Tampilkan Data" << endl;
    cout << "8. Exit" << endl;
    cout << "Pilih menu: ";
    cin >> choice;

    switch (choice) {
        case 1: {
            string namaProduk;
            int harga;
            cout << "Masukkan nama produk: ";
            cin >> namaProduk;
            cout << "Masukkan harga: ";
            cin >> harga;
            list.insertBelakang(namaProduk, harga);
            break;
        }
    }
```

```
case 2: {  
    string namaProduk;  
        cout << "Masukkan nama produk yang akan dihapus: ";  
    cin >> namaProduk;  
    list.hapusData(namaProduk);  
    break;  
}  
case 3: {  
    string namaProduk, namaProdukBaru;  
    int hargaBaru;  
    cout << "Masukkan nama produk yang akan diupdate: ";  
    cin >> namaProduk;  
    cout << "Masukkan nama produk baru: ";  
    cin >> namaProdukBaru;  
    cout << "Masukkan harga baru: ";  
    cin >> hargaBaru;  
    list.updateData(namaProduk, namaProdukBaru, hargaBaru);  
    break;  
}  
case 4: {  
    string namaProduk, namaProdukSebelum;  
    int harga;  
    cout << "Masukkan nama produk baru: ";  
    cin >> namaProduk;  
    cout << "Masukkan harga: ";  
    cin >> harga;  
    cout << "Masukkan nama produk sebelumnya: ";  
    cin >> namaProdukSebelum;  
    list.insertTengah(namaProduk, harga, namaProdukSebelum);  
    break;  
}
```

```
case 5: {  
    int posisi;  
    cout << "Masukkan posisi data yang akan dihapus: ";  
    cin >> posisi;  
    list.hapusDataUrutan(posisi);  
    break;  
}  
case 6: {  
    list.hapusSeluruhData();  
    break;  
}  
case 7: {  
    list.tampilData();  
    break;  
}  
case 8: {  
    cout << "Terima kasih!" << endl;  
    break;  
}  
default: {  
    cout << "Pilihan tidak valid!" << endl;  
    break;  
}  
}  
} while (choice != 8);  
  
return 0;  
}
```

Screenshots output



```
"D:\kumpulan tugas\inguided 2 modul 3\bin\Debug\inguided 2 modul 3.exe"
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 menu: 1
Masukkan nama produk: azarine
Masukkan harga: 65000

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 menu: 2
Masukkan nama produk yang akan dihapus: wardah
Data wardah tidak ditemukan!

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 menu: 3
Masukkan nama produk yang akan diupdate: hanasui
Masukkan nama produk baru: cleora
Masukkan harga baru: 55000
Data hanasui tidak ditemukan!

Toko Skincare Purwokerto
1. Tambah Data
```

Deskripsi :

1. **Struktur Node:** Terdapat sebuah kelas **Node** yang merepresentasikan setiap elemen dalam linked list. Setiap node memiliki dua pointer, yaitu **prev** untuk menunjuk ke node sebelumnya, dan **next** untuk menunjuk ke node selanjutnya.
2. **Kelas Double Linked List:** Kelas **DoubleLinkedList** mengimplementasikan berbagai operasi untuk mengelola linked list, seperti penambahan, penghapusan, pembaruan, dan tampilan data.
3. **Operasi Insertion:**
 - **insertDepan():** Menambahkan sebuah node di awal linked list.

- **insertBelakang():** Menambahkan sebuah node di akhir linked list.
- **insertTengah():** Menambahkan sebuah node di tengah linked list setelah node dengan nama tertentu.

4. Operasi Deletion:

- **hapusData():** Menghapus node dengan nama produk tertentu dari linked list.
- **hapusDataUrutan():** Menghapus node pada posisi tertentu dalam linked list.

5. Operasi Update:

- **updateData():** Memperbarui nama produk dan harga dari node dengan nama produk tertentu.

6. Operasi Tampilan:

- **tampilData():** Menampilkan nama produk dan harganya dari setiap node dalam linked list.

7. Menu Utama:

- Program menyediakan sebuah menu yang memungkinkan pengguna untuk memilih operasi yang ingin dilakukan. Pengguna dapat memilih untuk menambah,

menghapus, memperbarui, atau menampilkan data dalam linked list. Pengguna juga dapat memilih untuk keluar dari program.

8. Pengecekan Kondisi:

- Program juga menyediakan pengecekan kondisi, seperti pengecekan apakah linked list kosong, apakah node yang dicari ada dalam linked list, dan lain sebagainya.

E. Referensi

<https://home.iitk.ac.in/~ynsingh/phd/10204070.pdf>