

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN  
ALGORITMA**

**MODUL III  
SINGLE AND DOUBLE LINKED  
LIST**



**Disusun Oleh :**

Raka Andriy Shevchenko  
2311102054

**Dosen**

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

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

## A. Dasar Teori

### 1. Single Linked List

Single linked list adalah jenis data structure yang terdiri dari node-node yang terhubung satu sama lain melalui pointer. Setiap node mengandung data dan pointer ke node berikutnya. Pada single linked list, setiap node hanya memiliki pointer ke node berikutnya, tidak ada pointer ke node sebelumnya.<sup>[1]</sup>

Implementasi single linked list menggunakan C++ dapat dilakukan dengan menggunakan sebuah struct yang menggabungkan data dan pointer ke node berikutnya. Contohnya:<sup>[2]</sup>

```
struct Node {  
    int data;  
    Node* next;  
};
```

Setiap node mengandung data dan pointer ke node berikutnya. Pointers ini digunakan untuk menghubungkan node-node dalam list.

Untuk membuat single linked list, kita dapat menggunakan class yang mengelompokkan fungsi-fungsi untuk mengelola node. Contohnya:<sup>[2]</sup>

```
class LinkedList {  
    private:  
        Node* head;  
    public:  
        LinkedList() {  
            head = NULL;  
        }  
        void insertNode(int data);  
        void printList();  
        void deleteNode(int value);  
};
```

Dalam kode di atas, kita memiliki class `LinkedList` yang mengelompokkan pointer ke node pertama (head) dan fungsi untuk mengelola node. Fungsi `insertNode` digunakan untuk menambahkan node baru ke list, `printList` digunakan untuk menampilkan semua node dalam list, dan `deleteNode` digunakan untuk menghapus node dengan nilai tertentu.

## 2. Double Linked List

Double linked list adalah jenis data structure yang terdiri dari node-node yang terhubung satu sama lain melalui pointer. Setiap node mengandung data dan pointer ke node berikutnya dan pointer ke node sebelumnya. Pada double linked list, setiap node memiliki pointer ke node berikutnya dan pointer ke node sebelumnya.<sup>[1]</sup>

Implementasi double linked list menggunakan C++ dapat dilakukan dengan menggunakan sebuah struct yang menggabungkan data dan pointer ke node berikutnya dan sebelumnya. Contohnya:<sup>[2]</sup>

```
struct Node {  
    int data;  
    Node* next;  
    Node* prev;  
};
```

Setiap node mengandung data, pointer ke node berikutnya (next), dan pointer ke node sebelumnya (prev). Pointers ini digunakan untuk menghubungkan node-node dalam list.

Untuk membuat double linked list, kita dapat menggunakan class yang mengelompokkan fungsi-fungsi untuk mengelola node. Contohnya:<sup>[2]</sup>

```
class DoubleLinkedList {  
    private:  
        Node* head;  
    public:  
        DoubleLinkedList() {
```

```
        head = NULL;
    }
    void insertNode(int data);
    void printList();
    void deleteNode(int value);
};
```

Dalam kode di atas, kita memiliki class `DoubleLinkedList` yang mengelompokkan pointer ke node pertama (`head`) dan fungsi untuk mengelola node. Fungsi `insertNode` digunakan untuk menambahkan node baru ke list, `printList` digunakan untuk menampilkan semua node dalam list, dan `deleteNode` digunakan untuk menghapus node dengan nilai tertentu.

## B. Guided

### a. Guided 1

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();
}

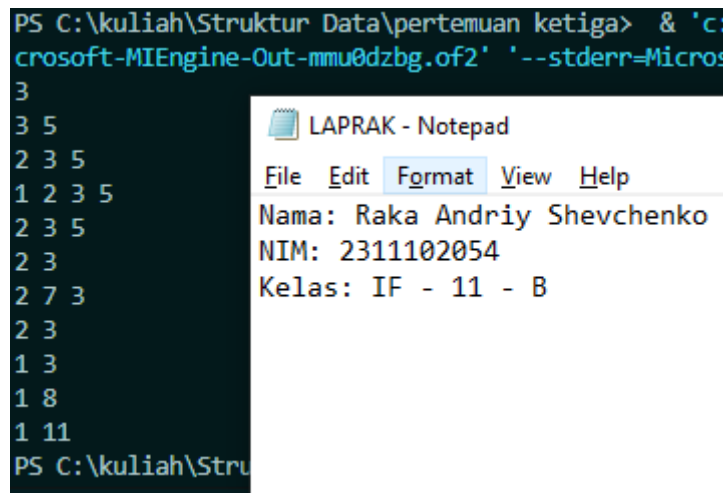
```

```

    return 0;
}

```

Output:



```

PS C:\kuliah\Struktur Data\pertemuan ketiga> & 'c:\Program Files\Microsoft Visual Studio\2019\Community\VC\Tools\MSI\14395b64-1841-4bbe-bb6c-32a6985491e1\Microsoft.VisualStudio.Setup.Common.exe --stderr=Microsoft.VisualStudio.Setup.Common'
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:\kuliah\Stru

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

Deskripsi:

Program tersebut adalah implementasi dari linked list tunggal (singly linked list) dalam bahasa C++. Program ini memulai dengan mendefinisikan sebuah struktur Node yang memiliki dua anggota, yaitu data untuk menyimpan nilai dari elemen dan next untuk menunjuk ke elemen berikutnya.

Program kemudian menyediakan sejumlah fungsi untuk operasi dasar pada linked list. Fungsi-fungsi ini termasuk penambahan elemen di depan (insertDepan) dan di belakang (insertBelakang), penghapusan elemen di depan (hapusDepan) dan di belakang (hapusBelakang), penghitungan jumlah elemen dalam list (hitungList), penambahan elemen di posisi tengah (insertTengah), penghapusan elemen di posisi tengah (hapusTengah), serta pengubahan nilai elemen di depan (ubahDepan), di belakang (ubahBelakang), dan di posisi tengah (ubahTengah).

Di dalam fungsi main(), program menguji setiap operasi pada linked list dengan memanggil fungsi-fungsi tersebut dan menampilkan isi linked list setelah setiap operasi. Ini memungkinkan penggunaan linked list dalam program untuk menyimpan dan mengelola data dengan berbagai cara sesuai kebutuhan.

b. Guided 2

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;
}

```

Output:

#### 1. Menu

```

PS C:\kuliah\Struktur Data\pertemuan ketiga> & 'c:\Users\crosoft-MIEngine-Out-wsxp34so.mqu' '--stderr=Microsoft-MI
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

#### 2. Add data

```

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

```

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

3. Delete data

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
```

```
LAPRAK - Notepad
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

4. Update data

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 1
Enter new data: 3
```

```
LAPRAK - Notepad
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

5. Clear data

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 4
```

```
LAPRAK - Notepad
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

6. Display data

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
3
```

```
LAPRAK - Notepad
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

Deskripsi:

Program di atas adalah implementasi dari Doubly Linked List menggunakan C++. Doubly Linked List adalah struktur data linear di mana setiap elemen terhubung dengan dua pointer, yaitu pointer ke elemen sebelumnya dan

pointer ke elemen berikutnya.

Di dalam program ini, terdapat dua kelas utama: Node yang merepresentasikan setiap elemen dalam Doubly Linked List, dan DoublyLinkedList yang merepresentasikan struktur data Doubly Linked List itu sendiri. Di dalam main() function, terdapat loop tak terbatas yang memungkinkan pengguna untuk memilih operasi apa yang ingin dilakukan pada Doubly Linked List. Ada enam pilihan yang tersedia:

Menambahkan data baru.

Menghapus data.

Memperbarui data.

Menghapus semua data.

Menampilkan semua data.

Keluar dari program.



## C. Unguided

### a. Unguided 1

Source Code:

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

// Deklarasi kelas Node untuk merepresentasikan simpul dalam
// linked list
class Node {
public:
    string nama; // Variabel untuk menyimpan nama mahasiswa
    int usia;    // Variabel untuk menyimpan usia mahasiswa
    Node* next;  // Pointer ke simpul berikutnya dalam linked
// list

    // Konstruktor untuk membuat objek Node baru dengan nama
// dan usia yang diberikan
    Node(string nama, int usia) {
        this->nama = nama;
        this->usia = usia;
        next = nullptr; // Pointer next diatur menjadi nullptr
// karena simpul baru akan ditambahkan ke akhir linked list
    }
};

// Deklarasi kelas LinkedList untuk menyimpan linked list dari
// objek Node
class LinkedList {
private:
    Node* head; // Pointer ke simpul pertama dalam linked
// list

public:
    // Konstruktor untuk menginisialisasi linked list dengan
// head yang menunjuk ke nullptr (linked list kosong)
    LinkedList() {
        head = nullptr;
    }

    // Metode untuk menambahkan mahasiswa baru ke akhir linked
// list
```

```

void tambahMahasiswa(string nama, int usia) {
    // Membuat simpul baru dengan nama dan usia yang
    diberikan
    Node* new_node = new Node(nama, usia);
    // Jika linked list kosong, maka head diatur menjadi
    simpul baru
    if (head == nullptr) {
        head = new_node;
        return;
    }
    // Jika linked list tidak kosong, maka mencari simpul
    terakhir dalam linked list
    Node* last_node = head;
    while (last_node->next != nullptr) {
        last_node = last_node->next;
    }
    // Menambahkan simpul baru sebagai simpul berikutnya
    dari simpul terakhir
    last_node->next = new_node;
}

// Metode untuk menampilkan data mahasiswa dalam linked
list
void tampilkanMahasiswa() {
    Node* current_node = head;
    while (current_node != nullptr) {
        cout << current_node->nama << "\t" <<
current_node->usia << endl;
        current_node = current_node->next;
    }
}

// Metode untuk menghapus mahasiswa dari linked list
berdasarkan nama
void hapusMahasiswa(string nama) {
    Node* current_node = head;
    Node* prev_node = nullptr;

    // Mencari mahasiswa dengan nama yang diberikan
    while (current_node != nullptr) {
        if (current_node->nama == nama) {
            // Jika ditemukan, menghapus simpul dari
            linked list

```

```

        if (prev_node == nullptr) {
            head = current_node->next;
        } else {
            prev_node->next = current_node->next;
        }
        delete current_node;
        cout << "Mahasiswa dengan nama '" << nama <<
"" berhasil dihapus." << endl;
        return;
    }
    prev_node = current_node;
    current_node = current_node->next;
}

// Jika tidak ditemukan, memberi pesan bahwa mahasiswa
tidak ditemukan dalam linked list
    cout << "Mahasiswa dengan nama '" << nama << "' tidak
ditemukan dalam linked list." << endl;
}

// Metode untuk menambahkan mahasiswa baru di depan linked
list
void addInFront(string nama, int usia) {
    // Membuat simpul baru dengan nama dan usia yang
diberikan
    Node* new_node = new Node(nama, usia);
    // Menyambungkan simpul baru ke simpul pertama,
kemudian membuat simpul baru sebagai head
    new_node->next = head;
    head = new_node;
    cout << "Mahasiswa dengan nama '" << nama << "'
berhasil ditambahkan di depan linked list." << endl;
}

// Metode untuk menambahkan mahasiswa baru di antara dua
mahasiswa
void addBetween(string nama_sebelum, string nama_baru, int
usia_baru) {
    Node* current_node = head;
    // Mencari simpul dengan nama_sebelum dalam linked
list
    while (current_node != nullptr) {
        if (current_node->nama == nama_sebelum) {

```

```

        // Jika ditemukan, membuat simpul baru dengan
nama_baru dan usia_baru
        Node* new_node = new Node(nama_baru,
usia_baru);
        // Menyambungkan simpul baru ke simpul yang
berikutnya dari simpul saat ini
        new_node->next = current_node->next;
        // Menyambungkan simpul saat ini ke simpul
baru
        current_node->next = new_node;
        cout << "Mahasiswa dengan nama '" << nama_baru
<< "' berhasil ditambahkan setelah '" << nama_sebelum << "'."
<< endl;

        return;
    }
    current_node = current_node->next;
}
// Jika nama_sebelum tidak ditemukan dalam linked
list, memberi pesan bahwa nama_sebelum tidak ditemukan
    cout << "Mahasiswa dengan nama '" << nama_sebelum <<
"" tidak ditemukan dalam linked list." << endl;
}

// Metode untuk memperbarui informasi mahasiswa
berdasarkan nama
void update(string nama, string nama_baru, int usia_baru)
{
    Node* current_node = head;

    // Mencari mahasiswa dengan nama yang diberikan
    while (current_node != nullptr) {
        if (current_node->nama == nama) {
            // Jika ditemukan, memperbarui nama dan usia
mahasiswa
            current_node->nama = nama_baru;
            current_node->usia = usia_baru;
            cout << "Informasi mahasiswa dengan nama '" <<
nama << "' berhasil diperbarui." << endl;
            return;
        }
        current_node = current_node->next;
    }
}

```

```

        // Jika tidak ditemukan, memberi pesan bahwa mahasiswa
        tidak ditemukan dalam linked list
        cout << "Mahasiswa dengan nama '" << nama << "' tidak
        ditemukan dalam linked list." << endl;
    }

    // Metode untuk mendapatkan pointer ke simpul pertama
    dalam linked list (head)
    Node* getHead() {
        return head;
    }
};

// Fungsi utama
int main() {
    LinkedList linkedList;

    // Loop untuk menampilkan menu dan menerima input dari
    pengguna
    while (true) {
        cout << "\nMenu:" << endl;
        cout << "1. Tambah Mahasiswa di Akhir" << endl;
        cout << "2. Tampilkan Mahasiswa" << endl;
        cout << "3. Hapus Mahasiswa" << endl;
        cout << "4. Tambah Mahasiswa di Tengah" << endl;
        cout << "5. Tambah Mahasiswa di Depan" << endl;
        cout << "6. Update Informasi Mahasiswa" << endl;
        cout << "7. Keluar" << endl;

        int pilihan;
        cout << "Pilih menu: ";
        cin >> pilihan;

        // Switch case untuk memilih operasi yang diinginkan
        oleh pengguna
        switch (pilihan) {
            case 1: {
                string nama;
                int usia;
                cout << "Masukkan nama mahasiswa: ";
                cin >> nama;
                cout << "Masukkan usia mahasiswa: ";
                cin >> usia;
            }
        }
    }
}

```

```

        linkedList.tambahMahasiswa(nama, usia);
        cout << "Mahasiswa berhasil ditambahkan di akhir."
<< endl;
        break;
    }
    case 2: {
        // Menampilkan daftar mahasiswa jika linked list
        tidak kosong, jika kosong memberi pesan
        if (linkedList.getHead() == nullptr) {
            cout << "Linked list kosong." << endl;
        } else {
            cout << "Nama\tUsia" << endl;
            linkedList.tampilkanMahasiswa();
        }
        break;
    }
    case 3: {
        string nama;
        cout << "Masukkan nama mahasiswa yang ingin
dihapus: ";
        cin >> nama;
        linkedList.hapusMahasiswa(nama);
        break;
    }
    case 4: {
        string nama_sebelum, nama_baru;
        int usia_baru;
        cout << "Masukkan nama mahasiswa sebelum
penambahan: ";
        cin >> nama_sebelum;
        cout << "Masukkan nama mahasiswa baru: ";
        cin >> nama_baru;
        cout << "Masukkan usia mahasiswa baru: ";
        cin >> usia_baru;
        linkedList.addBetween(nama_sebelum, nama_baru,
usia_baru);
        break;
    }
    case 5: {
        string nama;
        int usia;
        cout << "Masukkan nama mahasiswa: ";
        cin >> nama;

```

```

        cout << "Masukkan usia mahasiswa: ";
        cin >> usia;
        linkedList.addInFront(nama, usia);
        break;
    }
    case 6: {
        string nama, nama_baru;
        int usia_baru;
        cout << "Masukkan nama mahasiswa yang ingin
diperbarui: ";
        cin >> nama;
        cout << "Masukkan nama mahasiswa baru: ";
        cin >> nama_baru;
        cout << "Masukkan usia mahasiswa baru: ";
        cin >> usia_baru;
        linkedList.update(nama, nama_baru, usia_baru);
        break;
    }
    case 7:
        // Keluar dari program jika dipilih
        cout << "Program selesai." << endl;
        return 0;
    default:
        // Memberi pesan jika pilihan tidak valid
        cout << "Menu tidak valid. Silakan pilih menu yang
sesuai." << endl;
    }
}

return 0;
}

```

Output:

a. Data awal

Nama	Usia
Raka	18
John	19
Jane	20
Michael	18
Yusuke	19
Akechi	20
Hoshino	18
Karin	18

LAPRAK - Notepad

File Edit Format View Help

Nama: Raka Andriy Shevchenko

NIM: 2311102054

Kelas: IF - 11 - B

- b. Hapus data Akechi

```
Pilih menu: 3
Masukkan nama mahasiswa yang ingin dihapus: Akechi
Mahasiswa dengan nama 'Akechi' berhasil dihapus.
```

```
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

- c. Futaba

```
Masukkan nama mahasiswa sebelum penambahan: John
Masukkan nama mahasiswa baru: Futaba
Masukkan usia mahasiswa baru: 18
Mahasiswa dengan nama 'Futaba' berhasil ditambahkan setelah 'John'.
```

```
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

- d. Igor

```
Masukkan nama mahasiswa: Igor
Masukkan usia mahasiswa: 20
Mahasiswa dengan nama 'Igor' berhasil ditambahkan di depan linked list.
```

```
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

- e. Michael menjadi Reyn

```
Masukkan nama mahasiswa yang ingin diperbarui: Michael
Masukkan nama mahasiswa baru: Reyn
Masukkan usia mahasiswa baru: 18
Informasi mahasiswa dengan nama 'Michael' berhasil diperbarui.
```

```
File Edit Format View Help
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

- f. Data akhir

Nama	Usia
Igor	20
Raka	18
John	19
Futaba	18
Jane	20
Reyn	18
Yusuke	19
Hoshino	18
Karin	18

LAPRAK - Notepad

File Edit Format View Help

```
Nama: Raka Andriy Shevchenko
NIM: 2311102054
Kelas: IF - 11 - B
```

Deskripsi:

Program tersebut adalah implementasi sederhana dari aplikasi manajemen data mahasiswa menggunakan linked list dalam bahasa C++. Program ini memanfaatkan dua kelas utama: Node untuk merepresentasikan setiap elemen dalam linked list, dan LinkedList untuk mengelola linked list dan operasinya. Kelas Node memiliki tiga anggota data: nama untuk menyimpan nama mahasiswa, usia untuk menyimpan usia mahasiswa, dan next untuk menunjuk ke simpul berikutnya dalam linked list. Konstruktor kelas Node digunakan



untuk membuat objek Node baru dengan nama dan usia yang diberikan. Kelas LinkedList digunakan untuk menyimpan linked list dari objek Node. Ini memiliki metode untuk operasi dasar pada linked list seperti menambah, menampilkan, menghapus, dan memperbarui informasi mahasiswa. Metode tambahMahasiswa digunakan untuk menambahkan mahasiswa baru ke akhir linked list, tampilkanMahasiswa untuk menampilkan semua data mahasiswa dalam linked list, hapusMahasiswa untuk menghapus mahasiswa berdasarkan nama, addInFront untuk menambahkan mahasiswa di depan linked list, addBetween untuk menambahkan mahasiswa di antara dua mahasiswa, dan update untuk memperbarui informasi mahasiswa berdasarkan nama.

b. Unguided 2

Source Code:

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

// Struktur untuk menyimpan nama produk dan harganya
struct Product {
    string name;
    int price;
};

class Node {
public:
    Product data; // Menggunakan struktur Product
    Node* prev;
    Node* next;
};

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

    // Konstruktor untuk menginisialisasi head dan tail
    // menjadi nullptr
    DoublyLinkedList() {
        head = nullptr;
    }
};
```

```

        tail = nullptr;
    }

    // Fungsi untuk menambahkan data ke depan linked list
    void push(string name, int price) { // Menggunakan
string dan int untuk nama produk dan harga
        Node* newNode = new Node;
        newNode->data.name = name; // Memasukkan nama
produk
        newNode->data.price = price; // Memasukkan harga
        newNode->prev = nullptr;
        newNode->next = head;

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

        head = newNode;
    }

    // Fungsi untuk menghapus data dari depan linked list
    void pop() {
        if (head == nullptr) {
            return;
        }
        Node* temp = head;
        head = head->next;

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

        delete temp;
    }

    // Fungsi untuk memperbarui data produk berdasarkan
nama produk yang lama
    bool update(string oldName, string newName, int
newPrice) {

```

```

    Node* current = head;

    while (current != nullptr) {
        if (current->data.name == oldName) {
            current->data.name = newName;
            current->data.price = newPrice;
            return true;
        }
        current = current->next;
    }
    return false;
}

// Fungsi untuk menghapus semua data dari linked list
void deleteAll() {
    Node* current = head;
    while (current != nullptr) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    head = nullptr;
    tail = nullptr;
}

// Fungsi untuk menampilkan semua data produk dalam
// linked list
void display() {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data.name << "\t" << "\t" <<
current->data.price << endl; // Menampilkan nama dan harga
        current = current->next;
    }
    cout << endl;
}

// Fungsi untuk menyisipkan data pada posisi tertentu
// dalam linked list
void insert(string name, int price, int position) {
    Node* newNode = new Node;
    newNode->data.name = name;
    newNode->data.price = price;

```

```

        if (position <= 1) { // Insert at the beginning
            newNode->prev = nullptr;
            newNode->next = head;

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

            head = newNode;
        } else {
            Node* current = head;
            int currentPosition = 1;

            while (currentPosition < position - 1 &&
current != nullptr) {
                current = current->next;
                currentPosition++;
            }

            if (current == nullptr) { // Position out of
range
                cout << "Position out of range" << endl;
                delete newNode;
                return;
            }

            newNode->prev = current;
            newNode->next = current->next;

            if (current->next != nullptr) {
                current->next->prev = newNode;
            } else {
                tail = newNode;
            }

            current->next = newNode;
        }
    }
}

```

```

// Fungsi untuk menghapus data pada posisi tertentu
dalam linked list
void remove(int position) {
    if (head == nullptr) {
        return;
    }

    if (position <= 1) { // Remove the first node
        Node* temp = head;
        head = head->next;

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

        delete temp;
    } else {
        Node* current = head;
        int currentPosition = 1;

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

        if (current == nullptr || current->next ==
nullptr) { // Position out of range
            cout << "Position out of range" << endl;
            return;
        }

        Node* temp = current->next;
        current->next = temp->next;

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

```

```

        delete temp;
    }
}
};

int main() {
    DoublyLinkedList list;
    while (true) {
        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. Keluar" << endl;

        int choice;
        cout << "Masukkan pilihanmu: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                string name;
                int price;
                cout << "Enter product name: ";
                cin.ignore(); // Ignore previous newline
                character
                getline(cin, name); // Input nama produk
                cout << "Enter product price: ";
                cin >> price; // Input harga produk
                list.push(name, price);
                break;
            }
            case 2: {
                list.pop();
                break;
            }
            case 3: {
                string oldName, newName;
                int newPrice;
                cout << "Enter old product name: ";

```

```

        cin.ignore(); // Ignore previous newline
character
        getline(cin, oldName); // Input nama produk
yang lama
        cout << "Enter new product name: ";
        getline(cin, newName); // Input nama produk
yang baru
        cout << "Enter new product price: ";
        cin >> newPrice; // Input harga produk yang
baru
        bool updated = list.update(oldName,
newName, newPrice);
        if (!updated) {
            cout << "Product not found" << endl;
        }
        break;
    }
    case 4: {
        string name;
        int price, position;
        cout << "Enter product name: ";
        cin.ignore(); // Ignore previous newline
character
        getline(cin, name); // Input nama produk
        cout << "Enter product price: ";
        cin >> price; // Input harga produk
        cout << "Enter position to insert: ";
        cin >> position;
        list.insert(name, price, position);

        break;
    }
    case 5: {
        int position;
        cout << "Enter position to remove: ";
        cin >> position;
        list.remove(position);
        break;
    }
    case 6: {
        list.deleteAll();
        break;
    }
}

```

```

        case 7: {
            cout << "Nama Produk\ttHarga" << endl;
            list.display();
            break;
        }
        case 8: {
            return 0;
        }
        default: {
            cout << "Invalid choice" << endl;
            break;
        }
    }
}
return 0;
}

```

Output:

1. Data awal

Nama Produk	Harga
Originote	60000
Somethinc	150000
Skintific	100000
Wardah	50000
Hanasui	30000

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

2. Azarine

Enter product name: Azarine  
Enter product price: 65000  
Enter position to insert: 3

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

3. Hapus Wardah

5. Hapus data urutan tertentu  
6. Hapus seluruh data  
7. Tampilkan data  
8. Keluar  
Masukkan pilihanmu: 5  
Enter position to remove: 4

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B

4. Cleora

Enter old product name: Hanasui  
Enter new product name: Cleora  
Enter new product price: 55000

Nama: Raka Andriy Shevchenko  
NIM: 2311102054  
Kelas: IF - 11 - B



## 5. Menu

```
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. Keluar
Masukkan pilihanmu: █
```

## 6. Hasil akhir

```
Masukkan pilihanmu: 7
Nama Produk      Harga
Originote        60000
Somethinc        150000
Azarine          65000
Skintific        100000
Cleora           55000
```

Deskripsi:

Karena penjelasan detail sudah ada di source code, maka saya akan menjelaskan alur programnya saja. Pertama-tama, kode mendefinisikan sebuah struktur Product yang menyimpan nama dan harga produk. Kemudian, terdapat kelas Node yang merepresentasikan setiap simpul dalam Doubly Linked List. Setiap simpul memiliki dua pointer: prev yang menunjuk ke simpul sebelumnya, dan next yang menunjuk ke simpul berikutnya. Setiap simpul juga memiliki data bertipe Product untuk menyimpan informasi produk.

Kelas DoublyLinkedList adalah kelas yang digunakan untuk mengelola Doubly Linked List. Program menyediakan beberapa operasi dasar seperti push untuk menambah data di depan, pop untuk menghapus data dari depan, update untuk memperbarui data produk berdasarkan nama, deleteAll untuk menghapus seluruh data, dan lain-lain.

Di dalam fungsi main(), program memberikan menu kepada pengguna untuk memilih operasi yang ingin dilakukan. Pengguna dapat memilih

untuk menambah data produk, menghapus data, memperbarui data, menambah data pada posisi tertentu, menghapus data pada posisi tertentu, menghapus seluruh data, atau menampilkan seluruh data produk yang tersimpan. Program akan terus berjalan hingga pengguna memilih untuk keluar dari program.

#### D. Kesimpulan

Single linked list dan double linked list merupakan jenis data structure yang terdiri dari node-node yang terhubung satu sama lain melalui pointer. Single linked list hanya memiliki pointer ke node berikutnya, sedangkan double linked list memiliki pointer ke node berikutnya dan sebelumnya. Implementasi single linked list dan double linked list menggunakan C++ menggunakan sebuah struct yang menggabungkan data dan pointer ke node berikutnya dan sebelumnya. Kedua jenis linked list dapat digunakan untuk mengelola data yang terhubung satu sama lain melalui pointer, menambahkan, mengubah, dan menghapus node dengan mudah. Dengan menggunakan single linked list dan C++, kita dapat mengelola data yang terhubung satu sama lain melalui pointer, menambahkan, mengubah, dan menghapus node dengan mudah. Dengan menggunakan double linked list dan C++, kita dapat mengelola data yang terhubung satu sama lain melalui pointer, menambahkan, mengubah, dan menghapus node dengan mudah. Pilihan antara single linked list dan double linked list tergantung pada kebutuhan aplikasi dan tipe data yang akan digunakan.

#### E. Referensi

[1] Mahir Koding. Struktur Data – Single Linked List dengan Bahasa C. Diakses dari

<https://www.mahirkoding.com/struktur-data-single-linked-list-dengan-bahasa-c/>

[2] BitDegree. How to Use Linked List in C++. Diakses dari

<https://www.bitdegree.org/learn/linked-list-c-plus-plus>