

**LAPORAN PRAKTIKUM  
STRUKTUR DATA DAN ALGORITMA**

**MODUL III  
SINGLE LINKED LIST DAN DOUBLE LINKED LIST**



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

**Disusun oleh:**

**MUHAMMAD AULIA MUZZAKI NUGRAHA (2311102051)**

**IF-11-B**

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

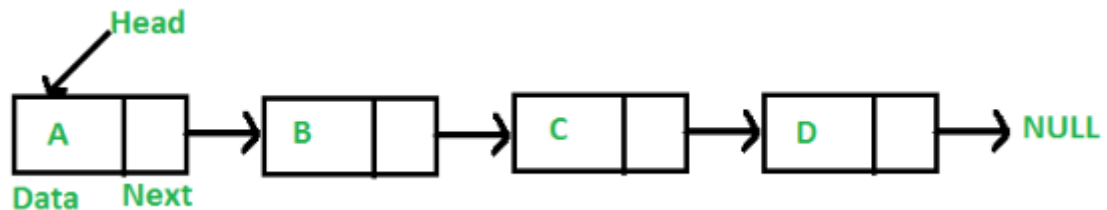
**2024**

# BAB I

## DASAR TEORI

### A. SINGLE LINKED LIST

Linked List adalah struktur data linear berbentuk rantai simpul dimana 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 dengan head atau simpul kepala. Apabila linked list berisi elemen kosong, maka nilai pointer dari head menunjukan ke NULL. Begitu juga untuk pointer berikutnya dari simpul terakhir atau simpul ekor akan menunjukan ke NULL.

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

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

### JENIS-JENIS LINKED LIST

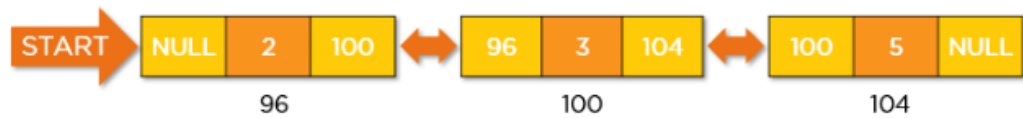
#### 1. Single Linked List

Single Linked List adalah list unidirectional. Jadi, hanya dapat melintasi dalam satu arah, yaitu dari simpul kepala ke simpul ekor.



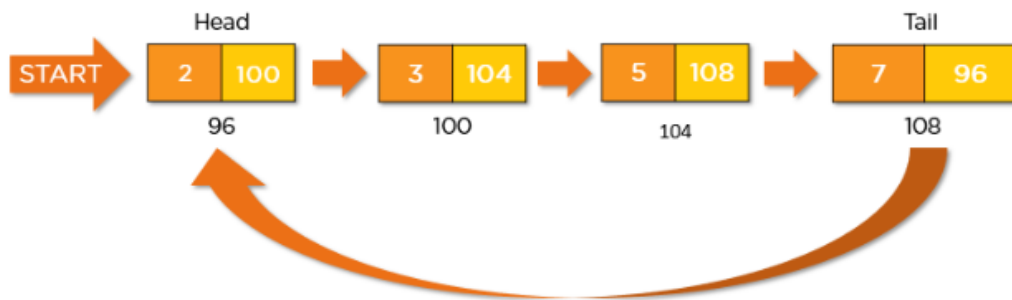
#### 2. Double linked list

Double linked list adalah linked list bidirectional. Jadi, kita hanya melintasi secara dua arah. Tidak seperti single linked list, simpul double linked list berisi satu pointer tambahan yang disebut previous pointer. Pointer ini menunjuk ke simpul sebelumnya.



### 3. Circular linked list

Circular linked list adalah linked list unidirectional. Kita hanya dapat melintasinya dalam satu arah. Tetapi jenis linked list memiliki simpul terakhir yang menunjuk ke simpul kepala. Jadi saat melintas, kita harus berhati-hati dan berhenti saat mengunjungi kembali simpul kepala.



### 4. Circular double linked list

Circular double linked list adalah gabungan dari Double linked list dan Circular linked list. Seperti Double linked list, linked list ini memiliki pointer tambahan yang disebut previous pointer, dan mirip dengan Circular linked list, simpul terakhirnya menunjuk pada simpul kepala. Jenis linked list ini adalah bidirectional. Jadi, kita bisa melintasinya dua arah.



## FUNGSI DAN KEGUNAAN LINKED LIST

Adapun fungsi dan kegunaan linked list adalah sebagai berikut :

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

- Membantu dalam manajemen memori.

Penerapan linked list banyak ditemui dalam beberapa kasus berikut :

- Linked list digunakan dalam penjadwalan Round-Robin untuk melacak giliran dalam permainan multi-pemain.
- Digunakan dalam aplikasi penampil gambar. Gambar sebelumnya dan berikutnya ditautkan, sehingga dapat diakses oleh tombol prev dan next.
- Dalam playlist musik, lagu yang sedang diputar ditautkan ke lagu sebelumnya dan berikutnya.

#### **KELEBIHAN LINKED LIST :**

- **Struktur data dinamis:** Linked list adalah himpunan dinamis sehingga dapat bertambah dan menyusut saat runtime dengan mengalokasikan dan membatalkan alokasi memori. Jadi kita tidak perlu memberikan ukuran awal dari linked list.
- **Tidak boros memori:** Dalam linked list, pemanfaatan memori yang efisien dapat dicapai karena ukuran linked list bertambah atau berkurang pada runtime sehingga tidak ada pemborosan memori dan tidak perlu mengalokasikan memori sebelumnya.
- **Implementasi:** Struktur data linier seperti stack dan queue seringkali mudah diimplementasikan menggunakan linked list.
- **Operasi penyisipan dan penghapusan:** Operasi penyisipan dan penghapusan cukup mudah dalam linked list. Kita tidak perlu menggeser elemen setelah operasi penyisipan atau penghapusan elemen, hanya alamat yang ada di pointer berikutnya saja yang perlu diperbarui.

#### **KELEMAHAN LINKED LIST :**

- **Penggunaan memori:** Linked list memerlukan lebih banyak memori dibandingkan dengan array. Karena dalam linked list, pointer juga perlu menyimpan alamat elemen berikutnya dan membutuhkan memori tambahan untuk dirinya sendiri.

- **Traversal:** Dalam traversal, linked list lebih banyak memakan waktu dibandingkan dengan array. Akses langsung ke elemen tidak bisa dilakukan pada linked list seperti array yang dapat akses elemen berdasarkan indeks. Untuk mengakses sebuah simpul pada posisi  $n$  dari linked list, kita harus melintasi semua simpul sebelumnya.
- **Reverse Traversing:** Dalam single linked list, reverse traversing tidak dimungkinkan, tetapi dalam kasus double-linked list, ini dapat dimungkinkan karena berisi pointer ke node yang terhubung sebelumnya dengan setiap node. Untuk melakukannya, diperlukan memori tambahan untuk pointer sebelumnya sehingga ada pemborosan memori.
- **Akses Acak:** Akses acak tidak bisa dilakukan dalam linked list karena alokasi memorinya yang dinamis.

## BAB II

### GUIDED

#### LATIHAN – GUIDED

##### 1. Guided 1

Latihan single linked list.

##### Source Code

```
#include <iostream>

using namespace std;

// Deklarasi Struct Node
struct Node {
    int data;
    Node* next;
};

Node* head;
Node* tail;

// Inisialisasi Node
void init() {
    head = NULL;
    tail = NULL;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == NULL;
}

// Tambah Node di depan
void insertDepan(int nilai) {
    Node* baru = new Node;
```

```

    baru->data = nilai;
    baru->next = NULL;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

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

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

```

```

// Tambah Node di posisi tengah
void insertTengah(int data, int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi diluar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node* baru = new Node();
        baru->data = data;
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1) {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Node di depan
void hapusDepan() {
    if (!isEmpty()) {
        Node* hapus = head;
        if (head->next != NULL) {
            head = head->next;
            delete hapus;
        } else {
            head = tail = NULL;
            delete hapus;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

```



```

}

// Hapus Node di belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head != tail) {
            Node* hapus = tail;
            Node* bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            tail = bantu;
            tail->next = NULL;
            delete hapus;
        } else {
            head = tail = NULL;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Node di posisi tengah
void hapusTengah(int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi diluar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node* hapus;
        Node* bantu = head;
        for (int nomor = 1; nomor < posisi - 1; nomor++) {
            bantu = bantu->next;
        }
    }
}

```

```

        hapus = bantu->next;
        bantu->next = hapus->next;
        delete hapus;
    }
}

// Ubah data Node di depan
void ubahDepan(int data) {
    if (!isEmpty()) {
        head->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di posisi tengah
void ubahTengah(int data, int posisi) {
    if (!isEmpty()) {
        if (posisi < 1 || posisi > hitungList()) {
            cout << "Posisi di luar jangkauan" << endl;
        } else if (posisi == 1) {
            cout << "Posisi bukan posisi tengah" << endl;
        } else {
            Node* bantu = head;
            for (int nomor = 1; nomor < posisi; nomor++) {
                bantu = bantu->next;
            }
            bantu->data = data;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

```

```

// Ubah data Node di belakang
void ubahBelakang(int data) {
    if (!isEmpty()) {
        tail->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

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

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

```

```

int main() {
    init();

    insertDepan(3); tampil();

    insertBelakang(5); tampil();

    insertDepan(2); tampil();

    insertDepan(1); tampil();

    hapusDepan(); tampil();

    hapusBelakang(); tampil();

    insertTengah(7, 2); tampil();

    hapusTengah(2); tampil();

    ubahDepan(1); tampil();

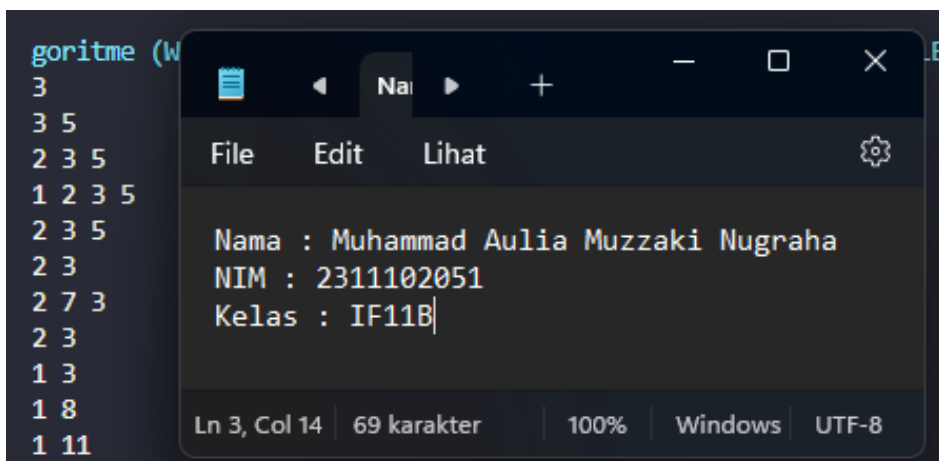
    ubahBelakang(8); tampil();

    ubahTengah(11, 2); tampil();

    return 0;
}

```

### Screenshoot program



### Deskripsi program

Program ini menerapkan struktur data linked list menggunakan struct Node. Fungsionalitas program mencakup inisialisasi list, verifikasi list kosong, penambahan data di bagian depan dan belakang, perhitungan jumlah data dalam list, penambahan data di posisi tengah, penghapusan data di bagian depan, belakang, dan posisi tengah, pengubahan data di bagian depan, belakang, dan posisi tengah, serta penghapusan semua data dalam list dan penampilan semua data dalam list.

## 2. Guided 2

### Latihan Double Linked list

#### Source code

```
#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

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

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

    void push(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

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

```

        head = newNode;
    }

void pop() {
    if (head == nullptr) {
        return;
    }
    Node* temp = head;
    head = head->next;

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

    delete temp;
}

bool update(int oldData, int newData) {
    Node* current = head;

    while (current != nullptr) {
        if (current->data == oldData) {
            current->data = newData;
            return true;
        }
        current = current->next;
    }
    return false;
}

void deleteAll() {

```

```

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

    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;
    }
}

```

```
switch (choice) {  
    case 1: {  
        int data;  
        cout << "Enter data to add: ";  
        cin >> data;  
        list.push(data);  
        break;  
    }  
    case 2: {  
        list.pop();  
        break;  
    }  
    case 3: {  
        int oldData, newData;  
        cout << "Enter old data: ";  
        cin >> oldData;  
        cout << "Enter new data: ";  
        cin >> newData;  
        bool updated = list.update(oldData, newData);  
        if (!updated) {  
            cout << "Data not found" << endl;  
        }  
        break;  
    }  
    case 4: {  
        list.deleteAll();  
        break;  
    }  
    case 5: {  
        list.display();  
        break;  
    }  
    case 6: {  
        return 0;  
    }  
}
```

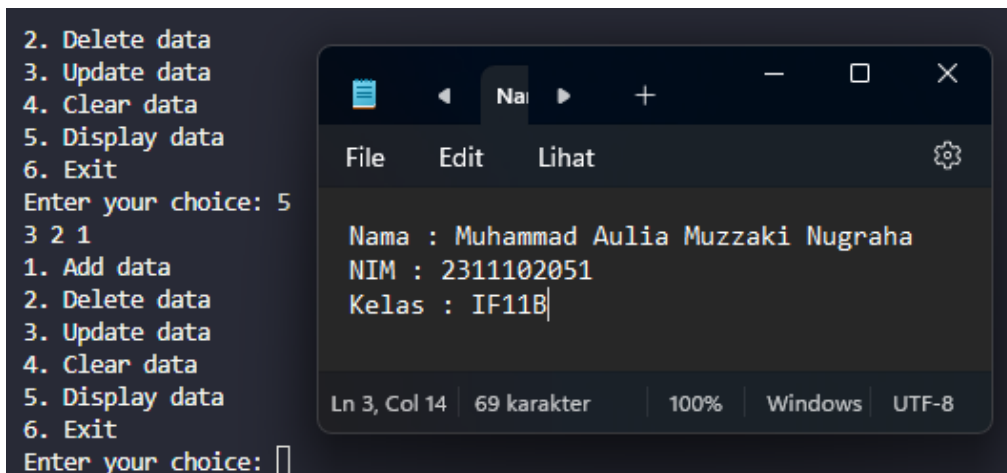


```

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

```

### Screenshoot program



```

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

```

Nama : Muhammad Aulia Muzzaki Nugraha  
 NIM : 2311102051  
 Kelas : IF11B

Ln 3, Col 14 | 69 karakter | 100% | Windows | UTF-8

### Deskripsi program

Double Linked List adalah struktur data linier yang terdiri dari node-node yang saling berhubungan, dimana setiap node memiliki dua pointer: sebuah pointer ke node sebelumnya (prev) dan sebuah pointer ke node berikutnya (next). Kelas Node digunakan untuk mendefinisikan struktur setiap node. Kelas ini memiliki atribut data yang menyimpan nilai dan dua pointer (sebelumnya dan berikutnya). Kelas DoublyLinkedList adalah implementasi dari daftar tertaut itu sendiri, dengan penunjuk awal dan akhir sebagai penanda awal dan akhir daftar tertaut.

Dalam fungsi main(), program menyediakan opsi operasional berikut yang dapat dilakukan pada daftar tertaut ganda: Menambah data, menghapus data, mengubah data, menghapus semua data, menampilkan data, keluar dari program, dll. Dalam opsi memanggil metode yang sesuai pada objek DoublyLinkedList sesuai dengan operasi yang diminta oleh pengguna. Misalnya, jika Anda memilih untuk menambahkan data,

program akan meminta pengguna untuk memasukkan data baru dan memanggil metode `push()` untuk menambahkan data ke daftar tertaut. Seluruh proses ini berulang hingga pengguna keluar dari program.

## BAB III

### UNGUIDED

#### TUGAS – UNGUIDED

##### 1. Unguided 1

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

##### Source Code

```
#include <iostream>

using namespace std;

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

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    // Function to insert data at the beginning of the list
    void insertAtBeginning(string nama, int usia) {
        Node* newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = head;
        head = newNode;
    }
}
```

```

// Function to insert data at the end of the list
void insertAtEnd(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;
}

// Function to insert data after a specific node
void insertAfter(string nama, int usia, string keyNama) {
    Node* temp = head;
    while (temp != nullptr && temp->nama != keyNama) {
        temp = temp->next;
    }
    if (temp == nullptr) {
        cout << "Data dengan nama " << keyNama << " tidak ditemukan.\n";
        return;
    }

    Node* newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;

```

```

        newNode->next = temp->next;

        temp->next = newNode;
    }

    // Function to delete a node with given nama
    void deleteNode(string nama) {
        Node* temp = head;
        Node* prev = nullptr;

        if (temp != nullptr && temp->nama == nama) {
            head = temp->next;
            delete temp;
            return;
        }

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

        if (temp == nullptr) {
            cout << "Data dengan nama " << nama << " tidak
ditemukan.\n";
            return;
        }

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

    // Function to display all data in the list
    void display() {
        Node* temp = head;
        cout << "Nama\tUsia\n";
    }

```

```

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

int main() {
    LinkedList list;

    list.insertAtBeginning("Rayya", 19);
    list.insertAtEnd("John", 19);
    list.insertAtEnd("Jane", 20);
    list.insertAtEnd("Michael", 18);
    list.insertAtEnd("Yusuke", 19);
    list.insertAtEnd("Akechi", 20);
    list.insertAtEnd("Hoshino", 18);
    list.insertAtEnd("Karin", 18);

    cout << "Data awal:\n";
    list.display();

    cout << "\nHapus data Akechi:\n";
    list.deleteNode("Akechi");
    list.display();

    cout << "\nTambahkan data Futaba setelah John:\n";
    list.insertAfter("Futaba", 18, "John");
    list.display();

    cout << "\nTambahkan data Igor di awal:\n";
    list.insertAtBeginning("Igor", 20);
    list.display();

    cout << "\nUbah data Michael menjadi Reyn:\n";

```

```

list.deleteNode("Michael");

list.insertAtEnd("Reyn", 18);

list.display();

return 0;
}

```

## Screenshoot program

The screenshot shows a C++ program with a linked list. The initial state is as follows:

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

The program prompts for deletion: "Hapus data Akechi:". The output shows the list after deletion:

Nama	Usia
Rayya	19
John	19
Jane	20
Michael	18
Yusuke	19
Hoshino	18
Karin	18

The program then prompts for insertion: "Tambahkan data Futaba setelah John:". The output shows the list after insertion:

Nama	Usia
Rayya	19
John	19
Futaba	18
Jane	20
Michael	18
Yusuke	19
Hoshino	18
Karin	18

The screenshot shows a C++ program with a linked list. The initial state is as follows:

Nama	Usia
Igor	20
Rayya	19
John	19
Futaba	18
Jane	20
Michael	18
Yusuke	19
Hoshino	18
Karin	18

The program prompts for deletion: "Ubah data Michael menjadi reyn:". The output shows the list after deletion:

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

## Deskripsi program

Program ini mengimplementasikan linked list yang menyimpan data siswa yaitu nama dan umur. Setiap node dalam daftar tertaut diwakili oleh struktur node dengan dua atribut, nama dan umur, serta penunjuk berikutnya ke node berikutnya. Kelas LinkedList memiliki berbagai fungsi untuk operasi dasar daftar tertaut termasuk: Misalnya, menyisipkan data di awal (insertAtBeginning), di akhir (insertAtEnd), setelah node tertentu (insertAfter), atau menghapus node dengan nama tertentu (deleteNode). Fungsi Display menampilkan semua data dalam daftar tertaut. Fungsi utama melakukan beberapa operasi untuk mendemonstrasikan penggunaan daftar tertaut ini, termasuk menambah, menghapus, dan membaca data.

Pada tahap awal, data mahasiswa dimasukkan ke dalam linked list menggunakan berbagai fungsi penambahan yang telah didefinisikan. Setelah itu, beberapa operasi dijalankan, seperti penghapusan data Akechi, penambahan data Futaba setelah John, penambahan data Igor di awal, dan pengubahan data Michael menjadi Reyn. Setiap operasi tersebut diikuti dengan menampilkan isi linked list untuk memperlihatkan hasil dari setiap operasi tersebut.

## 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.

### Source code

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

class Node {
public:
    string namaProduk;
    int harga;
    Node* prev;
    Node* next;
};
```



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

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

    void push(string namaProduk, int harga) {
        Node* newNode = new Node;
        newNode->namaProduk = namaProduk;
        newNode->harga = harga;
        newNode->prev = nullptr;
        newNode->next = head;

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

    void insertAfter(string namaProduk, int harga, string
keyNamaProduk) {
        Node* current = head;

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

        if (current == nullptr) {
```

```

        cout << "Data dengan nama produk " << keyNamaProduk <<
" tidak ditemukan.\n";

        return;

    }

    Node* newNode = new Node;
    newNode->namaProduk = namaProduk;
    newNode->harga = harga;
    newNode->prev = current;
    newNode->next = current->next;

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

    current->next = newNode;
}

void deleteNode(string namaProduk) {
    Node* current = head;

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

    if (current == nullptr) {
        cout << "Data dengan nama produk " << namaProduk << "
tidak ditemukan.\n";
        return;
    }

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

```

```

        head = current->next;

    }

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

    delete current;
}

bool update(string oldNamaProduk, string newNamaProduk, int
newHarga) {
    Node* current = head;

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

    if (current == nullptr) {
        cout << "Data dengan nama produk " << oldNamaProduk <<
" tidak ditemukan.\n";
        return false;
    }

    current->namaProduk = newNamaProduk;
    current->harga = newHarga;
    return true;
}

void deleteAll() {
    Node* current = head;

    while (current != nullptr) {
        Node* temp = current;

```

```

        current = current->next;

        delete temp;

    }

    head = nullptr;
    tail = nullptr;
}

void display() {
    Node* current = head;

    cout << left << setw(20) << "Nama Produk" << setw(10) <<
"Harga" << endl;

    while (current != nullptr) {
        cout << left << setw(20) << current->namaProduk <<
setw(10) << current->harga << endl;

        current = current->next;
    }

    cout << endl;
}

};

int main() {

    DoublyLinkedList list;

    list.push("Originote", 60000);
    list.push("Somethinc", 150000);
    list.push("Skintific", 100000);
    list.push("Wardah", 50000);
    list.push("Hanasui", 30000);

    while (true) {

        cout << "AUL STOREE" << endl;

        cout << "=====" << 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 << endl;

int choice;

cout << "Enter your choice: ";
cin >> choice;
cout << endl;

switch (choice) {
    case 1: {
        string namaProduk;
        int harga;
        cout << "Enter nama produk: ";
        cin >> namaProduk;
        cout << "Enter harga: ";
        cin >> harga;
        list.push(namaProduk, harga);
        break;
    }
    case 2: {
        string namaProduk;
        cout << "Enter nama produk yang akan dihapus: ";
        cin >> namaProduk;
        list.deleteNode(namaProduk);
        break;
    }
    case 3: {
        string oldNamaProduk, newNamaProduk;
        int newHarga;
```

```

        cout << "Enter nama produk yang akan diupdate: ";
        cin >> oldNamaProduk;
        cout << "Enter nama produk baru: ";
        cin >> newNamaProduk;
        cout << "Enter harga baru: ";
        cin >> newHarga;
        bool updated = list.update(oldNamaProduk,
newNamaProduk, newHarga);
        if (!updated) {
            cout << "Data not found" << endl;
        }
        break;
    }
    case 4: {
        string namaProduk, keyNamaProduk;
        int harga;
        cout << "Enter nama produk yang akan ditambahkan
setelahnya: ";
        cin >> keyNamaProduk;
        cout << "Enter nama produk baru: ";
        cin >> namaProduk;
        cout << "Enter harga: ";
        cin >> harga;
        list.insertAfter(namaProduk, harga, keyNamaProduk);
        break;
    }
    case 5: {
        string namaProduk;
        cout << "Enter nama produk yang akan dihapus: ";
        cin >> namaProduk;
        list.deleteNode(namaProduk);
        break;
    }
    case 6: {
        list.deleteAll();
    }

```

```
        cout << "Semua data telah dihapus." << endl;
        break;
    }
    case 7: {
        cout << "Data Produk:\n";
        list.display();
        break;
    }
    case 8: {
        return 0;
    }
    default: {
        cout << "Invalid choice" << endl;
        break;
    }
}

return 0;
}
```

**Screenshoot program**

```
AUL STOREE
=====
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

Enter your choice: 7

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

AUL STOREE
=====
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

Enter your choice: []
```

Na

File Edit Lihat

Nama : Muhammad Aulia Muzzaki Nugraha  
NIM : 2311102051  
Kelas : IF11B

Ln 3, Col 14 69 karakter 100% Window UTF-8

### Deskripsi program

Program ini digunakan untuk menambahkan data, hapus data, update data, tambahkan data urutan tertentu, hapus data urutan tertentu, hapus seluruh data, tampilkan data.

Dalam deklarasi Kelas 'node' digunakan untuk merepresentasikan produk skincare, setiap node memiliki atribut 'namaProduk' untuk menyimpan nama produk, 'harga' untuk menyimpan harga produk, serta dua pointer 'prev' dan 'next' untuk menunjukan ke node sebelumnya dan selanjutnya dalam linked list.

Deklarasi Kelas Double Linked List, Kelas 'DoublyLinkedList' adalah kelas utama yang bertanggung jawab atas manajemen linked list, Memiliki 'head' dan 'tail' yang masing-masing menunjukan ke node pertama dan terakhir dalam linked list.





## **BAB IV**

### **KESIMPULAN**

Dalam bahasa pemrograman C++, array adalah salah satu struktur data paling mendasar yang digunakan untuk menyimpan kumpulan data yang serupa dalam satu variabel. Setiap elemen dalam array diakses menggunakan indeks numerik yang dimulai dari 0, yang memudahkan akses data secara efisien. Penting untuk memperhatikan bahwa saat mendeklarasikan array, kita harus menentukan ukuran array yang sesuai dengan jumlah elemen yang akan disimpan di dalamnya. Ukuran array bersifat tetap dan tidak dapat diubah setelah deklarasi, oleh karena itu, perlu dipilih dengan hati-hati agar tidak terjadi pemborosan memori atau kekurangan memori yang dapat menyebabkan program menjadi tidak stabil.

Selain itu, array juga memungkinkan pemrosesan paralel dari data karena setiap elemen dapat diakses secara terpisah dan operasi dapat diterapkan pada setiap elemen secara independen. Namun, perlu diingat bahwa array memiliki beberapa keterbatasan, termasuk ukuran tetap dan sulitnya mengubah ukuran array setelah deklarasi. Oleh karena itu, perlu dipertimbangkan dengan hati-hati dalam penggunaannya, terutama dalam konteks aplikasi yang membutuhkan fleksibilitas dalam penyimpanan data.

## **DAFTAR PUSTAKA**

Assiten Praktikum. Modul 3 Single and Double Linked List. Learning Management System.  
Diakses pada 30 Maret 2024.

Trivusi web, Struktur data Linked list : Pengertian, karakteristik, dan jenis-jenisnya (2022)

<https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html>