

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL 3
SINGLE AND DOUBLE LINKED LIST**



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

**Disusun Oleh :
FALAH ASBI PANGESTU (2311102045)
IF-11-B**

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

A. Dasar Teori

Linked list adalah struktur data linear yang terdiri dari sejumlah simpul (node) yang terhubung secara sekuensial. Setiap simpul terdiri dari dua bagian, yaitu data yang disimpan dan alamat atau pointer yang menunjuk ke simpul berikutnya dalam urutan.

1. Single Linked List

Single linked list adalah jenis linked list di mana setiap simpul hanya memiliki satu pointer yang menunjuk ke simpul berikutnya dalam urutan. Ini adalah bentuk yang paling sederhana dari linked list.

- Kelebihan: Menggunakan memori dengan efisien karena hanya memerlukan satu pointer per simpul.
- Kekurangan: Traversal mundur (dari belakang ke depan) tidak mungkin dilakukan tanpa menggunakan struktur tambahan.
- Struktur simpul/node

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

Dalam struktur simpul di atas, data merupakan data yang akan disimpan dalam simpul, dan next adalah pointer yang menunjuk ke simpul berikutnya.

- Operasi dasar
 - Insertion (Penyisipan): Data dapat dimasukkan di awal, tengah, atau akhir linked list dengan mengubah pointer next.
 - Deletion (Penghapusan): Simpul dapat dihapus dengan mengatur ulang pointer next.
 - Traversal (Penelusuran): Linked list dapat diiterasi dari awal hingga akhir dengan menggunakan pointer next.
 - Searching (Pencarian): Pencarian data dapat dilakukan dengan menelusuri linked list hingga menemukan data yang sesuai.

2. Double Linked List

Double linked list adalah jenis linked list di mana setiap simpul memiliki dua pointer, yaitu pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya. Dengan demikian, traversal dari depan ke belakang dan sebaliknya dimungkinkan dalam double linked list.

- Kelebihan: Dapat melakukan traversal maju dan mundur dengan mudah karena setiap simpul memiliki pointer prev dan next.
- Kekurangan: Memerlukan lebih banyak ruang memori

karena setiap simpul memiliki dua pointer.

- Struktur simpul/node

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

Dalam struktur simpul di atas, data merupakan data yang akan disimpan dalam simpul, prev adalah pointer yang menunjuk ke simpul sebelumnya, dan next adalah pointer yang menunjuk ke simpul berikutnya.

- Operasi dasar
 - Insertion (Penyisipan): Data dapat dimasukkan di awal, tengah, atau akhir linked list dengan mengubah pointer next dan prev.
 - Deletion (Penghapusan): Simpul dapat dihapus dengan mengatur ulang pointer next dan prev.
 - Traversal (Penelusuran): Linked list dapat diiterasi dari awal hingga akhir, atau sebaliknya, menggunakan pointer next dan prev.
 - Searching (Pencarian): Pencarian data dapat dilakukan dengan menelusuri linked list hingga menemukan data yang sesuai.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

```
#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
bool isEmpty() {
    return head == NULL;
}

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

// Hitung Jumlah List
int hitungList() {
    Node *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) {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

```

```

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

// Hapus Tengah
void hapusTengah(int posisi) {
    if (posisi < 1 || posisi > hitungList()) {
        cout << "Posisi di luar jangkauan" << endl;
    } else if (posisi == 1) {
        cout << "Posisi bukan posisi tengah" << endl;
    } else {
        Node *hapus, *bantu, *bantu2;

```

```

        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()) {
        head->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah 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;
            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()) {
        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 = head;
    if (!isEmpty()) {
        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

PS D:\SEMESTER 2\STRUKDAT\MODUL-3> & 'c:\Users\ACER\.vscode\extensions\ms-vscode.cpptools\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-4a4t...rs4' '--stderr=Microsoft-MIEngine-Error-e15kngsh.2g5' '--pid=Micro...ucrt64\bin\g...

```

3
35
235
1235
235
23
273
23
13
18
111
PS D:\SEMESTER 2\STRUKDAT\MODUL-3>

```

Falah Asbi Pangestu
2311102045

Ln 9, Col 1 | 37 characters | 100% | Window | UTF-8

Deskripsi

1. Deklarasi Struct Node

- Struct `Node` digunakan untuk merepresentasikan simpul dalam linked list.
- Setiap simpul memiliki dua bagian, yaitu `data` untuk menyimpan nilai dan `next` untuk menunjuk ke simpul berikutnya.

2. Inisialisasi Linked List

Fungsi `init()` digunakan untuk menginisialisasi linked list dengan mengatur `head` dan `tail` menjadi NULL.

3. Pengecekan Kehampaan Linked List

- Fungsi `isEmpty()` digunakan untuk memeriksa apakah linked list kosong atau tidak.
- Mengembalikan true jika linked list kosong, dan false sebaliknya.

4. Penyisipan Data di Awal (`insertDepan`) dan di Akhir (`insertBelakang`) Linked List

- Fungsi `insertDepan()` dan `insertBelakang()` digunakan untuk menyisipkan data baru ke dalam linked list.
- Data baru akan ditambahkan sebagai simpul baru di awal atau di akhir linked list.

5. Hitung Jumlah Simpul (`hitungList`)

Fungsi `hitungList()` digunakan untuk menghitung jumlah simpul dalam linked list.

6. Penyisipan Data di Tengah Linked List (`insertTengah`)

Fungsi `insertTengah()` digunakan untuk menyisipkan data baru di antara simpul-simpul yang ada dalam linked list.

7. Penghapusan Data di Awal (`hapusDepan`) dan di Akhir (`hapusBelakang`) Linked List:

Fungsi `hapusDepan()` dan `hapusBelakang()` digunakan untuk menghapus simpul pertama atau terakhir dalam linked list.

8. Penghapusan Data di Tengah Linked List (`hapusTengah`)

Fungsi `hapusTengah()` digunakan untuk menghapus simpul di tengah linked list berdasarkan posisi yang ditentukan.

9. Perubahan Data pada Simpul (`ubahDepan`, `ubahBelakang`, `ubahTengah`)

Fungsi-fungsi ini digunakan untuk mengubah nilai data pada simpul tertentu di depan, di belakang, atau di tengah linked list.

10. Hapus Seluruh Data Linked List (`clearList`)

Fungsi `clearList()` digunakan untuk menghapus semua simpul dalam linked list dan mengatur `head` dan `tail` menjadi NULL.

11. Tampilkan Isi Linked List (`tampil`)

Fungsi `tampil()` digunakan untuk menampilkan isi dari seluruh linked list. Jika linked list kosong, akan dicetak pesan yang sesuai.

Guided 2

```
#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) {
            temp->next = nullptr;
        }
        delete temp;
    }
};
```

```

        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;
            }
        }
    }
}

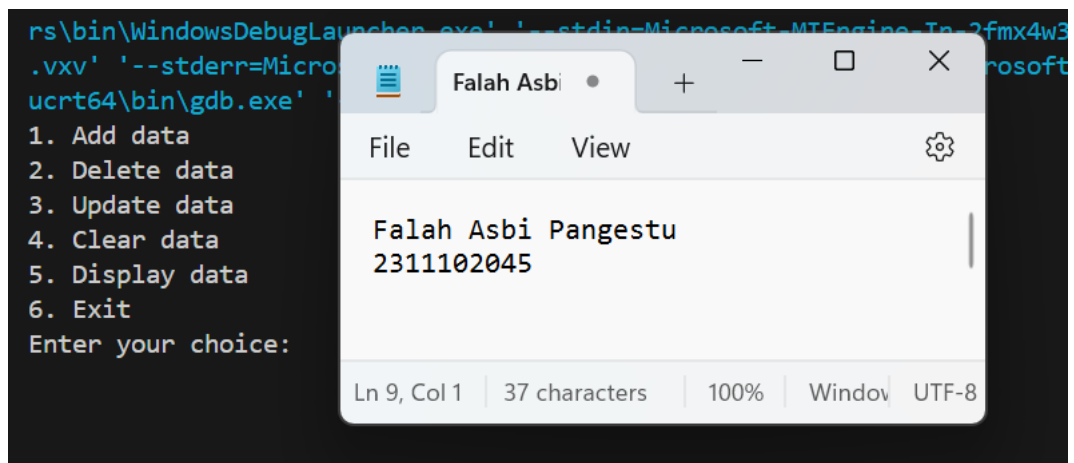
```

```

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

```

Screenshots Output



Deskripsi:

Doubly linked list adalah struktur data linier yang terdiri dari simpul-simpul yang saling terhubung, di mana setiap simpul memiliki dua pointer, yaitu pointer ke simpul sebelumnya (prev) dan pointer ke simpul berikutnya (next). Kelas Node digunakan untuk mendefinisikan struktur dari setiap simpul, yang memiliki atribut data untuk menyimpan nilai dan dua pointer prev dan next. Kelas DoublyLinkedList merupakan implementasi dari linked list itu sendiri, dengan memiliki pointer head dan tail sebagai penanda awal dan akhir dari linked list.

Dalam fungsi main(), program memberikan pilihan-pilihan operasi yang dapat dilakukan terhadap doubly linked list, seperti menambah data, menghapus data, mengubah data, menghapus seluruh data, menampilkan

data, atau keluar dari program. Setiap pilihan akan memanggil metode-metode yang sesuai pada objek DoublyLinkedList sesuai dengan operasi yang diminta oleh pengguna. Misalnya, jika memilih untuk menambah data, program akan meminta input data baru dari pengguna dan memanggil metode push() untuk menambahkan data tersebut ke linked list. Seluruh operasi ini akan terus berulang hingga pengguna memilih untuk keluar dari program.

- C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

```
#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";

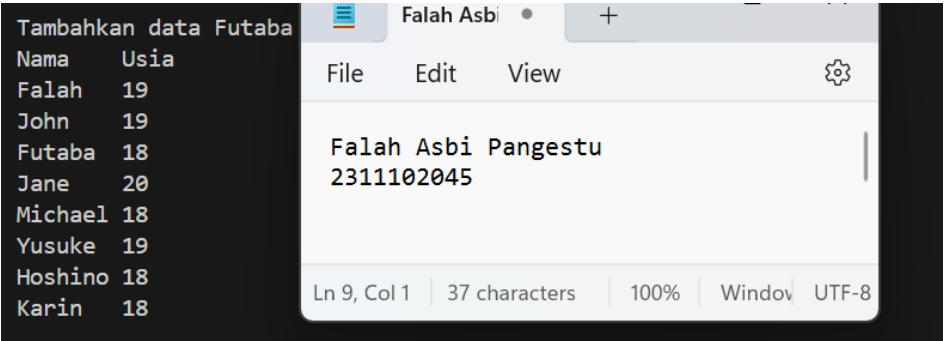
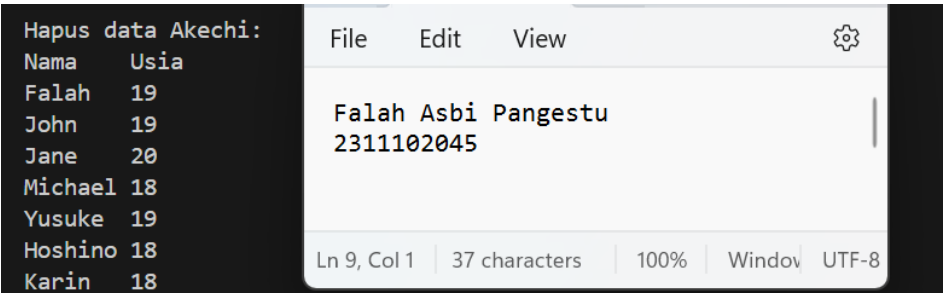
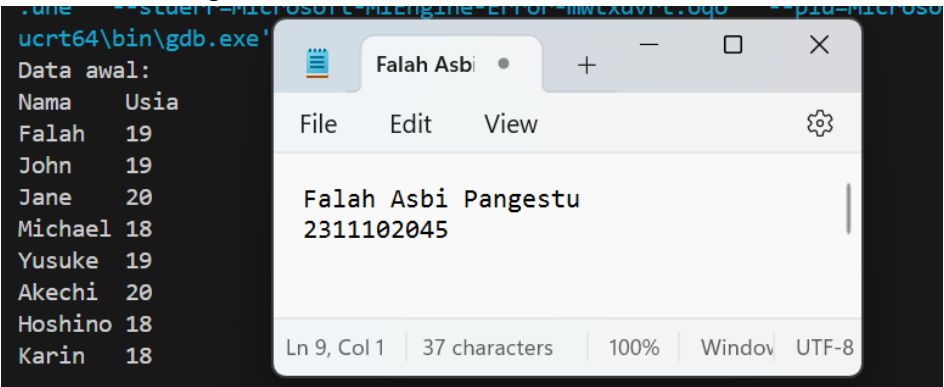
```

```
list.insertAtBeginning("Igor", 20);
list.display();

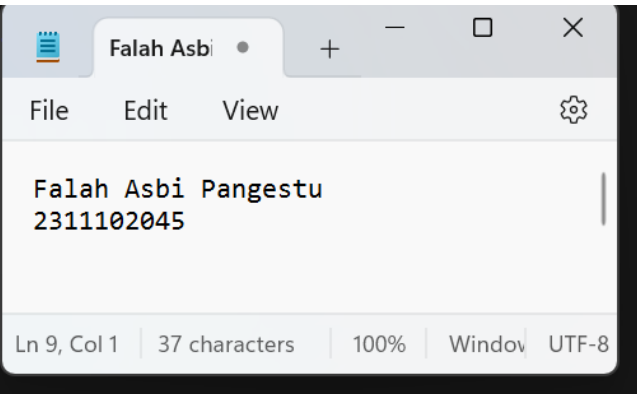
cout << "\nUbah data Michael menjadi Reyn:\n";
list.deleteNode("Michael");
list.insertAtEnd("Reyn", 18);
list.display();

return 0;
}
```

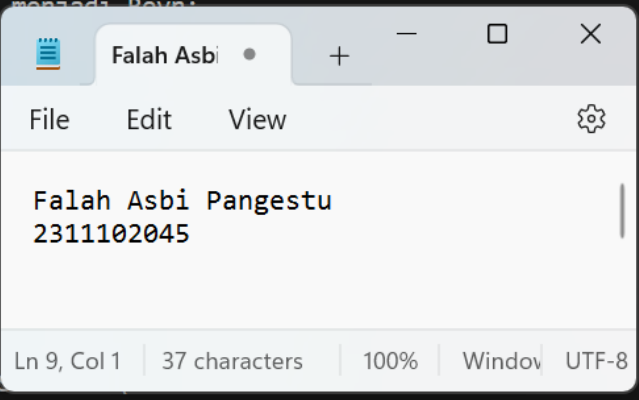
Screenshots Output



```
Tambahkan data Igor
Nama  Usia
Igor  20
Falah 19
John  19
Futaba 18
Jane  20
Michael 18
Yusuke 19
Hoshino 18
Karin  18
```



```
Ubah data Michael menjadi Reyn
Nama  Usia
Igor  20
Falah 19
John  19
Futaba 18
Jane  20
Yusuke 19
Hoshino 18
Karin  18
Reyn  18
PS D:\SEMESTER 2\
```



Deskripsi:

Program mengimplementasikan sebuah linked list yang menyimpan data mahasiswa, yaitu nama dan usia. Setiap simpul dalam linked list direpresentasikan oleh struktur Node, yang memiliki dua atribut yaitu nama dan usia, serta pointer next untuk menunjuk ke simpul berikutnya. Kelas LinkedList memiliki berbagai fungsi untuk operasi-operasi dasar pada linked list, seperti menyisipkan data di awal (insertAtBeginning), di akhir (insertAtEnd), setelah simpul tertentu (insertAfter), serta menghapus simpul dengan nama tertentu (deleteNode). Fungsi display digunakan untuk menampilkan seluruh data yang ada dalam linked list. Pada fungsi main, beberapa operasi seperti penambahan, penghapusan, dan pembacaan data dilakukan untuk mengilustrasikan penggunaan dari linked list ini.

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.

Unguided 2

```
#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 << "Toko Skincare Purwokerto" << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
        cout << "3. Update Data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
        cout << "5. Hapus Data Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data" << endl;
        cout << "7. Tampilkan Data" << endl;
        cout << "8. Exit" << endl;
        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;
}

```

Screenshots Output

```

ucrt64\bin\gdb.exe' '--interpreter=mi'
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

Enter your choice: 7

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

```

Notepad window content:

```

Falah Asbi Pangestu
2311102045

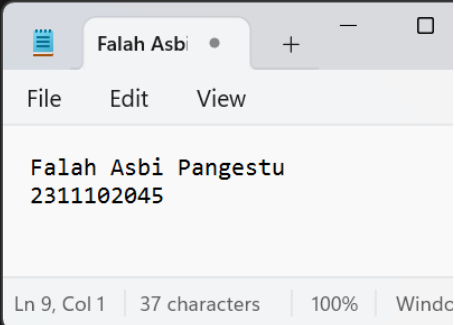
```

Notepad status bar: Ln 9, Col 1 | 37 characters | 100%

```
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

Enter your choice: 4


Enter nama produk yang akan ditambahkan setelahnya: Skintific
Enter nama produk baru: Azarine
Enter 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

Enter your choice: 2

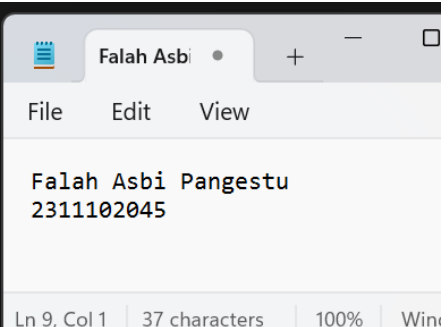
Enter nama produk yang akan dihapus: Wardah
```



```
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

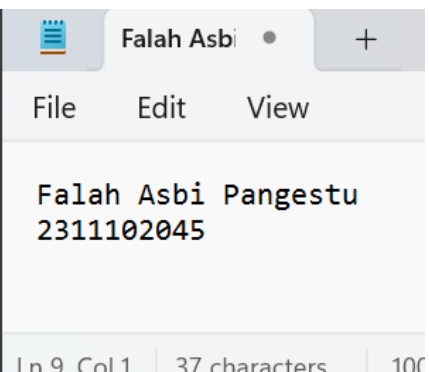
Enter your choice: 3

Enter nama produk yang akan diupdate: Hanasui
Enter nama produk baru: Cleora
Enter harga baru: 55000
```



```
Enter your choice: 7

Data Produk:
Nama Produk      Harga
Cleora            55000
Skintific         100000
Azarine           65000
Somethinc         150000
Originote         60000
```



The image shows a Notepad window with a single line of text: "Falalah Asbi Pangestu 2311102045". The window title is "Falalah Asbi". The status bar at the bottom indicates "Ln 9, Col 1" and "37 characters".

Deskripsi:

1. Deklarasi Struktur `Node`: Struktur `Node` digunakan untuk merepresentasikan setiap elemen dalam linked list. Setiap `Node` memiliki dua atribut, yaitu `namaProduk` yang menyimpan nama produk, `harga` yang menyimpan harga produk, serta dua pointer `prev` dan `next` untuk menunjukkan ke `Node` sebelumnya dan selanjutnya dalam linked list.
2. Deklarasi `DoublyLinkedList`: Kelas `DoublyLinkedList` merupakan implementasi dari linked list ganda (doubly linked list). Kelas ini memiliki pointer `head` dan `tail` sebagai penanda awal dan akhir dari linked list.
3. Fungsi `push`: Fungsi ini digunakan untuk menambahkan data baru ke depan linked list. Data baru berupa nama produk dan harga.
4. Fungsi `insertAfter`: Fungsi ini memungkinkan penambahan data baru setelah node tertentu dalam linked list. Data baru juga terdiri

dari nama produk dan harga.

5. Fungsi ``deleteNode``: Fungsi ini digunakan untuk menghapus node dengan nama produk tertentu dari linked list.
6. Fungsi ``update``: Fungsi ini memungkinkan pengubahan nama produk dan harga dari suatu node dengan nama produk tertentu.
7. Fungsi ``deleteAll``: Fungsi ini menghapus seluruh data dari linked list.
8. Fungsi ``display``: Fungsi ini menampilkan seluruh data yang ada dalam linked list, dengan format nama produk dan harga yang diatur secara rapi menggunakan ``setw`` dari ``<iomanip>``.
9. Fungsi ``main``: Fungsi utama dari program. Di dalamnya, pengguna diberikan pilihan operasi untuk melakukan tindakan seperti menambahkan, menghapus, memperbarui, atau menampilkan data produk. Program akan terus berjalan hingga pengguna memilih untuk keluar. Setiap operasi diimplementasikan menggunakan metode-metode yang sudah didefinisikan dalam kelas ``DoublyLinkedList``.

D. Kesimpulan

- Konsep Dasar: Linked list adalah struktur data linear yang terdiri dari serangkaian elemen, yang setiap elemennya terhubung dengan elemen berikutnya melalui pointer. Setiap elemen dalam linked list disebut node.
- Single Linked List: Single linked list adalah jenis linked list di mana setiap node memiliki satu pointer yang menunjuk ke node berikutnya. Ini adalah implementasi paling sederhana dari linked list.
- Double Linked List: Double linked list adalah jenis linked list di mana setiap node memiliki dua pointer, yaitu satu yang menunjuk ke node sebelumnya dan satu lagi yang menunjuk ke node berikutnya. Ini memungkinkan traversal maju dan mundur dalam linked list dengan mudah.
- Alokasi Dinamis: Linked list memungkinkan alokasi memori secara dinamis, yang berarti ruang memori baru dapat dialokasikan atau didealokasikan saat diperlukan, tanpa perlu mengalokasikan ruang yang telah ditentukan sebelumnya.
- Penyisipan dan Penghapusan: Salah satu keunggulan utama dari linked list adalah kemampuannya untuk melakukan operasi penyisipan dan penghapusan dengan efisien, terutama jika operasi dilakukan di tengah-tengah linked list.

- Pencarian: Pencarian dalam linked list membutuhkan traversal dari awal atau akhir linked list hingga elemen yang dicari ditemukan. Hal ini membuat waktu pencarian bergantung pada panjang linked list.
- Kekurangan: Salah satu kelemahan utama dari linked list adalah kinerja akses acak yang buruk. Karena setiap elemen harus diakses secara berurutan, pencarian dengan indeks atau akses langsung ke elemen tertentu tidak seefisien array.
- Kelebihan: Meskipun memiliki beberapa kekurangan, linked list tetap berguna dalam beberapa kasus, terutama ketika operasi penyisipan dan penghapusan sering terjadi, atau ketika ukuran data dapat berubah secara dinamis.

E. Referensi

Jurusan Teknik Elektro, Fakultas Teknik. (2018). Pengenalan Bahasa C++, Algoritma Pemrograman, Integrated Development Equipment (IDE) Visual C++ dan Dasar-Dasar Bahasa C++. Semester Genap 2017/2018.

Stroustrup, B. (2022). A Tour of C++ (3rd Edition). Addison-Wesley. ISBN-10: 0-13-681648-7, ISBN-13: 978-0-13-681648-5. Retrieved from Stroustrup's Publications.