

**LAPORAN PRAKTIKUM STRUKTUR
DATA DAN ALGORITMA**

**MODUL III
SINGLE AND DOUBLE LINKED LIST**



Disusun Oleh :

NAMA : RIFKY DWI MAHARDIKA

NIM : 2311102043

Dosen

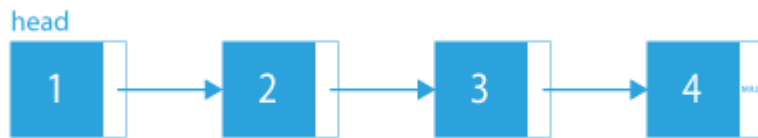
Wahyu Andi Saputra, S.Pd., M.Eng

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

A. DASAR TEORI

a. Single Linked List

Single Linked List adalah struktur data linier yang terdiri dari serangkaian node. Setiap node dalam Single Linked List berisi dua komponen: elemen data dan pointer yang menunjuk ke node berikutnya dalam urutan. Node pertama dalam barisan disebut head, dan node terakhir disebut tail. Jika penunjuk suatu simpul adalah nol, berarti simpul tersebut adalah simpul terakhir dalam urutan tersebut.



Untuk melintasi Single Linked List, Anda mulai dari kepala dan ikuti petunjuk hingga Anda mencapai akhir daftar. Memasukkan node baru ke dalam Single Linked List melibatkan pembaruan penunjuk dari node sebelumnya untuk menunjuk ke node baru, dan memperbarui penunjuk dari node baru untuk menunjuk ke node berikutnya. Menghapus sebuah node melibatkan memperbarui penunjuk dari node sebelumnya untuk menunjuk ke node berikutnya, dan membebaskan memori dari node yang dihapus.

b. Double Linked List

Double Linked List adalah jenis struktur data yang terdiri dari urutan node, masing-masing berisi dua penunjuk, bukan satu seperti pada Single Linked List. Setiap node dalam Double Linked List memiliki tiga komponen: elemen data, penunjuk ke node berikutnya, dan penunjuk ke node sebelumnya. Penunjuk ke node sebelumnya inilah yang membuat Double Linked List berbeda dari Single Linked List.



Memasukkan node baru ke dalam Double Linked List melibatkan pembaruan penunjuk dari node sebelumnya dan berikutnya agar menunjuk ke node baru, dan memperbarui penunjuk dari node baru agar menunjuk ke node sebelumnya dan berikutnya. Menghapus sebuah node melibatkan pembaruan pointer dari node sebelumnya dan berikutnya untuk melewati node yang dihapus, dan membebaskan memori dari node yang dihapus.

B. GUIDED

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 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();

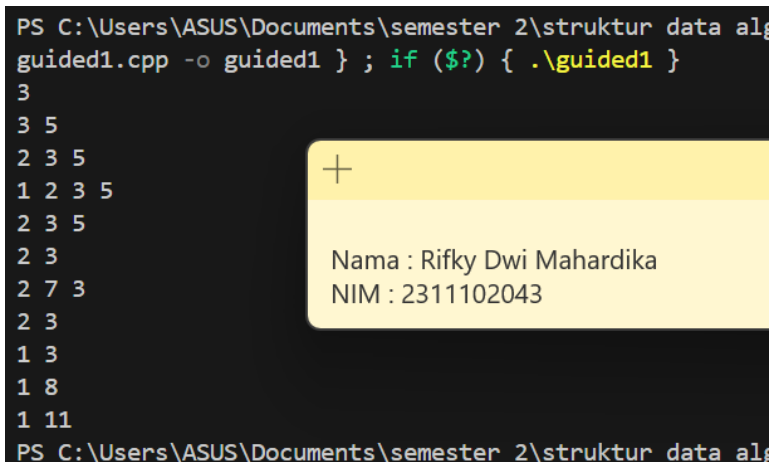
```

```

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;
}

```

Screenshots Output



```

PS C:\Users\ASUS\Documents\semester 2\struktur data alg
guided1.cpp -o guided1 } ; if ($?) { .\guided1 }
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:\Users\ASUS\Documents\semester 2\struktur data al

```

+
 Nama : Rifky Dwi Mahardika
 NIM : 2311102043

Deskripsi :

Program C++ di atas adalah implementasi dari struktur data linked list. Ini adalah sebuah program sederhana yang menampilkan operasi-operasi dasar pada linked list,

seperti penambahan elemen di depan, di belakang, di tengah, penghapusan elemen di depan, di belakang, di tengah, serta pengubahan elemen. Semua fungsi diatas digunakan didalam fungsi main() untuk menjalankan berbagai operasi pada linked list seperti menambahkan, menghapus, mengubah, dan menampilkan data-data dalam linked list. Jadi ini adalah sebuah program sederhana yang mengimplementasikan operasi dasar pada sebuah linked list.

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)
    {
        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

```

PS C:\Users\ASUS\Documents\semester 2\struktur data algorit
tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?) {
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: 57
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: 35
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 57
Enter new data: 88

```

+

Nama : Rifky Dwi Mahardika
NIM : 2311102043

```

1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
88
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit

```

+

Nama : Rifky Dwi Mahardika
NIM : 2311102043

```
Enter your choice: 5
88
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 2
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 4
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 6
PS C:\Users\ASUS\Documents\semester 2\struktur data algo
```

Deskripsi :

Program di atas, program sederhana yang mengimplementasikan operasi dasar pada sebuah doubly linked list (daftar berantai ganda).

- Deklarasi Class Node: Mendefinisikan sebuah kelas Node yang memiliki tiga anggota, yaitu data untuk menyimpan nilai integer, prev untuk menunjukkan ke node sebelumnya dalam linked list, dan next untuk menunjukkan ke node selanjutnya dalam linked list.
- Deklarasi Class DoublyLinkedList: Mendefinisikan sebuah kelas DoublyLinkedList yang berisi anggota-anggota head dan tail, yaitu pointer yang menunjuk pada node pertama dan terakhir dalam linked list.
- Konstruktor DoublyLinkedList(): Menginisialisasi head dan tail menjadi nullptr (kosong) saat objek dari kelas DoublyLinkedList dibuat.
- Fungsi push(): Menambahkan node baru di depan linked list. Node baru tersebut akan menjadi head baru. Jika linked list tidak kosong, prev dari node pertama sebelumnya akan menunjuk pada node baru.
- Fungsi pop(): Menghapus node pertama dari linked list. Jika linked list kosong, tidak ada operasi yang dilakukan. Jika tidak, node pertama akan dihapus dan head

akan diupdate. Jika linked list hanya memiliki satu node, head dan tail akan menjadi nullptr.

- Fungsi update(): Mencari dan mengubah nilai data dari node yang memiliki nilai data tertentu. Jika data ditemukan, nilai data tersebut akan diubah menjadi nilai baru.
- Fungsi deleteAll(): Menghapus semua node dalam linked list dan mengembalikan head dan tail menjadi nullptr.
- Fungsi display(): Menampilkan semua data node dalam linked list.
- Fungsi main(): Berisi loop utama yang meminta pengguna untuk memilih operasi yang ingin dilakukan, seperti menambah data, menghapus data, mengubah data, menampilkan data, dan keluar dari program.

Dalam loop tersebut, setelah pengguna memilih operasi, program akan menjalankan fungsi yang sesuai dengan pilihan pengguna menggunakan switch-case. Jika pengguna memilih keluar (pilihan 6), program akan berakhir.

C. UNGUIDED

Unguided 1

```
#include <iostream>
using namespace std;
struct Node
{
    string nama;
    int usia;
    Node *next;
};
Node *head;
Node *tail;
void init()
{
    head = NULL;
    tail = NULL;
}
bool isEmpty()
{
    return head == NULL;
}
void insertDepan(string nama, int usia)
{
    Node *baru = new Node;
    baru->nama = nama;
    baru->usia = usia;
    baru->next = NULL;
    if (isEmpty())
    {
        head = tail = baru;
    }
    else
    {
        baru->next = head;
        head = baru;
    }
}
void insertBelakang(string nama, int usia)
{
    Node *baru = new Node;
    baru->nama = nama;
    baru->usia = usia;
    baru->next = NULL;
    if (isEmpty())
```

```

    {
        head = tail = baru;
    }
    else
    {
        tail->next = baru;
        tail = baru;
    }
}
int hitungList()
{
    Node *hitung = head;
    int jumlah = 0;
    while (hitung != NULL)
    {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}
void insertTengah(string nama, int usia, 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->nama = nama;
        baru->usia = usia;
        Node *bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1)
        {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

```

```

}
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;
    }
}

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;
    }
}
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;
    }
}
void ubahDepan(string nama, int usia)
{
    if (!isEmpty())
    {
        head->nama = nama;
        head->usia = usia;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
void ubahTengah(string nama, int usia, 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->nama = nama;
        bantu->usia = usia;
    }
}
else
{
    cout << "List masih kosong!" << endl;
}
}

void ubahBelakang(string nama, int usia)
{
    if (!isEmpty())
    {
        tail->nama = nama;
        tail->usia = usia;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

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

```

```

void tampil()
{
    if (!isEmpty())
    {
        Node *bantu = head;
        while (bantu != NULL)
        {
            cout << bantu->nama << " ";
            cout << bantu->usia << " , ";
            bantu = bantu->next;
        }
        cout << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

int main()
{
    init();
    int menu, usia, posisi;
    string nama;
    cout << "\n# Menu Linked List Mahasiswa #" << endl;
    do
    {
        cout << "\n 1. Insert Depan"
              << "\n 2. Insert Belakang"
              << "\n 3. Insert Tengah"
              << "\n 4. Hapus Depan"
              << "\n 5. Hapus Belakang"
              << "\n 6. Hapus Tengah"
              << "\n 7. Ubah Depan"
              << "\n 8. Ubah Belakang"
              << "\n 9. Ubah Tengah"
              << "\n 10. Tampilkan"
              << "\n 0. Keluar Program"
              << "\n Pilihan : ";

        cin >> menu;
        switch (menu)
        {
            case 1:
                cout << "Masukkan Nama : ";
                cin >> nama;
                cout << "Masukkan Usia : ";

```

```
        cin >> usia;
        insertDepan(nama, usia);
        cout << endl;
        tampil();
        break;
    case 2:
        cout << "Masukkan Nama : ";
        cin >> nama;
        cout << "Masukkan Usia : ";
        cin >> usia;
        insertBelakang(nama, usia);
        cout << endl;
        tampil();
        break;
    case 3:
        cout << "Masukkan Posisi : ";
        cin >> posisi;
        cout << "Masukkan Nama : ";
        cin >> nama;
        cout << "Masukkan Usia : ";
        cin >> usia;
        insertTengah(nama, usia, posisi);
        cout << endl;
        tampil();
        break;
    case 4:
        hapusDepan();
        cout << endl;
        tampil();
        break;
    case 5:
        hapusBelakang();
        cout << endl;
        tampil();
        break;
    case 6:
        cout << "Masukkan Posisi : ";
        cin >> posisi;
        hapusTengah(posisi);
        cout << endl;
        tampil();
        break;
    case 7:
        cout << "Masukkan Nama : ";
        cin >> nama;
```



```

        cout << "Masukkan Usia : ";
        cin >> usia;
        ubahDepan(nama, usia);
        cout << endl;
        tampil();
        break;
    case 8:
        cout << "Masukkan Nama : ";
        cin >> nama;
        cout << "Masukkan Usia : ";
        cin >> usia;
        ubahBelakang(nama, usia);
        cout << endl;
        tampil();
        break;
    case 9:
        cout << "Masukkan Posisi : ";
        cin >> posisi;
        cout << "Masukkan Nama : ";
        cin >> nama;
        cout << "Masukkan Usia : ";
        cin >> usia;
        ubahTengah(nama, usia, posisi);
        cout << endl;
        tampil();
        break;
    case 10:
        tampil();
        break;
    default:
        cout << "Pilihan Salah" << endl;
        break;
    }
} while (menu != 0);
return 0;
}

```

Screenshots Output

- a. Masukkan data sesuai urutan berikut. (Gunakan insert depan, belakang atau tengah). Data pertama yang dimasukkan adalah nama dan usia anda.

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 1
Masukkan Nama : Rifky
Masukkan Usia : 18

Rifky 18 , Karin 18 , Hoshino 18 , Akechi 20 , Yusuke 19 , Michael 18 , Jane 20 , John 19
```

- b. Hapus data Akechi

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 6
Masukkan Posisi : 4

Rifky 18 , Karin 18 , Hoshino 18 , Yusuke 19 , Michael 18 , Jane 20 , John 19
```

- c. Tambahkan data berikut diantara John dan Jane : Futaba 18

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 3
Masukkan Posisi : 7
Masukkan Nama : Futaba
Masukkan Usia : 18

Rifky 18 , Karin 18 , Hoshino 18 , Yusuke 19 , Michael 18 , Jane 20 , Futaba 18 , John 19 ,
```

d. Tambahkan data berikut diawal : Igor 20

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 1
Masukkan Nama : Igor
Masukkan Usia : 20

Igor 20 , Rifky 18 , Karin 18 , Hoshino 18 , Yusuke 19 , Michael 18 , Jane 20 , Futaba 18 , John 19 ,
```

e. Ubah data Michael menjadi : Reyn 18

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 9
Masukkan Posisi : 9
Masukkan Nama : Reyn
Masukkan Usia : 18

Igor 20 , Rifky 18 , Karin 18 , Hoshino 18 , Yusuke 19 , Jane 20 , Futaba 18 , John 19 , Reyn 18 ,
```

f. Tampilkan seluruh data

```
1. Insert Depan
2. Insert Belakang
3. Insert Tengah
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : 10
Igor 20 , Rifky 18 , Karin 18 , Hoshino 18 , Yusuke 19 , Jane 20 , Futaba 18 , John 19 , Reyn 18 ,
```

Deskripsi :

Program diatas, sebuah program yang mengelola data mahasiswa menggunakan struktur data linked list.

- Deklarasi Struct Node: Mendefinisikan struktur data Node yang memiliki tiga anggota, yaitu nama untuk menyimpan nama mahasiswa, usia untuk menyimpan

usia mahasiswa, dan next untuk menunjukkan ke node selanjutnya dalam linked list.

- Deklarasi Pointer Head dan Tail: Deklarasi dua pointer head dan tail yang menunjuk pada node pertama dan terakhir dalam linked list.
- Fungsi init(): Inisialisasi head dan tail menjadi NULL di awal program.
- Fungsi isEmpty(): Memeriksa apakah linked list kosong atau tidak dengan memeriksa apakah pointer head adalah NULL atau tidak.
- Fungsi insertDepan(): Menambahkan node baru di depan linked list dengan data nama dan usia. Jika linked list masih kosong, node baru menjadi head. Jika tidak, node baru akan menjadi head baru dan menunjuk pada node sebelumnya.
- Fungsi insertBelakang(): Menambahkan node baru di belakang linked list dengan data nama dan usia. Jika linked list masih kosong, node baru menjadi head dan tail. Jika tidak, node baru akan menjadi tail baru dan menunjuk pada node sebelumnya.
- Fungsi hitungList(): Menghitung jumlah node dalam linked list dengan mengiterasi dari head hingga tail dan menghitung setiap node.
- Fungsi insertTengah(): Menambahkan node baru pada posisi tertentu di tengah linked list dengan data nama dan usia. Jika posisi diluar jangkauan atau posisi bukan di tengah, pesan kesalahan akan ditampilkan.
- Fungsi hapusDepan(): Menghapus node pertama dari linked list.
- Fungsi hapusBelakang(): Menghapus node terakhir dari linked list.
- Fungsi hapusTengah(): Menghapus node pada posisi tertentu di tengah linked list. Jika posisi diluar jangkauan atau posisi bukan di tengah, pesan kesalahan akan ditampilkan.
- Fungsi ubahDepan(): Mengubah data (nama dan usia) pada node pertama.
- Fungsi ubahTengah(): Mengubah data (nama dan usia) pada node pada posisi tertentu di tengah linked list. Jika posisi diluar jangkauan atau posisi bukan di tengah, pesan kesalahan akan ditampilkan.
- Fungsi ubahBelakang(): Mengubah data (nama dan usia) pada node terakhir.
- Fungsi clearList(): Menghapus semua node dalam linked list.
- Fungsi tampil(): Menampilkan semua data (nama dan usia) node dalam linked list.

Pada Fungsi main() Berisi loop utama yang memberikan opsi kepada pengguna untuk melakukan operasi-operasi seperti menambah data, menghapus data, mengubah data, menampilkan data, dan keluar dari program. Setelah pengguna memilih operasi, program akan menjalankan fungsi yang sesuai dengan pilihan pengguna menggunakan switch-case. Program akan terus berjalan sampai pengguna memilih opsi 0 untuk keluar.

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 pushCenter(string namaProduk, int harga, int posisi)
{
    if (posisi < 0)
    {
        cout << "Posisi harus bernilai non-negatif." << endl;
        return;
    }

    Node *newNode = new Node;
    newNode->namaProduk = namaProduk;
    newNode->harga = harga;

    if (posisi == 0 || head == nullptr)
    {
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr)
        {
            head->prev = newNode;
        }
        else
        {
            tail = newNode;
        }
        head = newNode;
    }
    else
    {
        Node *temp = head;
        int count = 0;
        while (temp != nullptr && count < posisi)
        {
            temp = temp->next;
            count++;
        }

        if (temp == nullptr)
        {
            newNode->prev = tail;
            newNode->next = nullptr;
            tail->next = newNode;
            tail = newNode;
        }
    }
}

```

```

    }
    else
    {
        newNode->prev = temp->prev;
        newNode->next = temp;
        temp->prev->next = newNode;
        temp->prev = newNode;
    }
}
}

```

```

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

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

    delete temp;
}

```

```

void popCenter(int posisi)
{
    if (head == nullptr)
    {
        cout << "List kosong. Tidak ada yang bisa dihapus." << endl;
        return;
    }

    if (posisi < 0)
    {
        cout << "Posisi harus bernilai non-negatif." << endl;
        return;
    }
}

```



```

if (posisi == 0)
{
    Node *temp = head;
    head = head->next;

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

    delete temp;
}
else
{
    Node *temp = head;
    int count = 0;
    while (temp != nullptr && count < posisi)
    {
        temp = temp->next;
        count++;
    }

    if (temp == nullptr)
    {
        cout << "Posisi melebihi ukuran list. Tidak ada yang
dihapus." << endl;
        return;
    }

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

```

```

    }
}

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

    while (current != nullptr)
    {
        if (current->namaProduk == oldNamaProduk)
        {
            current->namaProduk = newNamaProduk;
            current->harga = newHarga;
            return true;
        }
        current = current->next;
    }
    return false;
}

bool updateCenter(string newNamaProduk, int newHarga, int posisi)
{
    if (head == nullptr)
    {
        cout << "List kosong. Tidak ada yang dapat diperbarui." <<
endl;
        return false;
    }

    if (posisi < 0)
    {
        cout << "Posisi harus bernilai non-negatif." << endl;
        return false;
    }

    Node *current = head;
    int count = 0;

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

```

```

        if (current == nullptr)
        {
            cout << "Posisi melebihi ukuran list. Tidak ada yang
diperbarui." << endl;
            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()
{
    if (head == nullptr)
    {
        cout << "List kosong." << endl;
        return;
    }

    Node *current = head;

    cout << setw(37) << setfill('-') << "-" << setfill(' ') << endl;
    cout << "| " << setw(20) << left << "Nama Produk"
        << " | " << setw(10) << "Harga"
        << " |" << endl;
    cout << setw(37) << setfill('-') << "-" << setfill(' ') << endl;

    while (current != nullptr)
    {

```

```

        cout << "|" << setw(20) << left << current->namaProduk << "
| " << setw(10) << current->harga << " |" << endl;
        current = current->next;
    }
    cout << setw(37) << setfill('-') << "-" << setfill(' ') << endl;
}
};

int main()
{
    DoublyLinkedList list;
    int choice;
    cout << endl
        << "#== Toko Skincare Purwokerto ==#" << endl;
    do
    {
        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 << "Pilihan : ";
        cin >> choice;

        switch (choice)
        {
            case 1:
            {
                string namaProduk;
                int harga;
                cout << "Masukkan nama produk: ";
                cin.ignore();
                getline(cin, namaProduk);
                cout << "Masukkan harga produk: ";
                cin >> harga;
                list.push(namaProduk, harga);
                break;
            }
            case 2:
            {
                list.pop();

```

```

        break;
    }
    case 3:
    {
        string newNamaProduk;
        int newHarga, posisi;
        cout << "Masukkan posisi produk: ";
        cin >> posisi;
        cout << "Masukkan nama baru produk: ";
        cin >> newNamaProduk;
        cout << "Masukkan harga baru produk: ";
        cin >> newHarga;
        bool updatedCenter = list.updateCenter(newNamaProduk,
newHarga, posisi);
        if (!updatedCenter)
        {
            cout << "Data not found" << endl;
        }
        break;
    }
    case 4:
    {
        string namaProduk;
        int harga, posisi;
        cout << "Masukkan posisi data produk: ";
        cin >> posisi;
        cout << "Masukkan nama produk: ";
        cin.ignore();
        getline(cin, namaProduk);
        cout << "Masukkan harga produk: ";
        cin >> harga;
        list.pushCenter(namaProduk, harga, posisi);
        break;
    }
    case 5:
    {
        int posisi;
        cout << "Masukkan posisi data produk: ";
        cin >> posisi;
        list.popCenter(posisi);
        break;
    }
    case 6:
    {
        list.deleteAll();
    }

```

```

        break;
    }
    case 7:
    {
        list.display();
        break;
    }
    case 8:
    {
        return 0;
    }
    default:
    {
        cout << "Invalid choice" << endl;
        break;
    }
}
} while (choice != 8);

return 0;
}

```

Screenshots Output

- a. Tambahkan produk Azarine dengan harga 65000 diantara Somethinc dan Skintific

```

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
Pilihan : 4
Masukkan posisi data produk: 2
Masukkan nama produk: Azarine
Masukkan harga produk: 65000

```

+

Nama : Rifky Dwi Mahardika
NIM : 2311102043

- b. Hapus produk wardah

```

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
Pilihan : 5
Masukkan posisi data produk: 4

```

+

Nama : Rifky Dwi Mahardika
NIM : 2311102043

- c. Update produk Hanasui menjadi Cleora dengan harga 55.000

```
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
Pilihan :
3
Masukkan posisi produk: 4
Masukkan nama baru produk: Cleora
Masukkan harga baru produk: 55000
```

- d. Tampilkan menu

```
PS C:\Users\ASUS\Documents\semester 2\struktur data algori
unguided2.cpp -o unguided2 } ; if ($?) { .\unguided2 }

#== 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
Pilihan : 
```

```
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
Pilihan : 7
```

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

Deskripsi :

Program tersebut, ialah program untuk mengelola data produk skincare menggunakan doubly linked list. Penjelasan code program tersebut, antara lain:

- Deklarasi Struct Node: Mendefinisikan struktur data Node yang memiliki empat anggota, yaitu namaProduk untuk menyimpan nama produk skincare, harga untuk menyimpan harga produk, prev untuk menunjukkan ke node sebelumnya dalam linked list, dan next untuk menunjukkan ke node selanjutnya dalam linked list.
- Deklarasi Kelas DoublyLinkedList: Mendefinisikan kelas DoublyLinkedList yang memiliki pointer head dan tail sebagai node pertama dan terakhir dalam linked list.
- Fungsi push(): Menambahkan node baru di depan linked list dengan data nama produk dan harga. Node baru akan menjadi head baru.
- Fungsi pushCenter(): Menambahkan node baru pada posisi tertentu di dalam linked list dengan data nama produk dan harga. Node baru akan diletakkan di posisi yang diinginkan.
- Fungsi pop(): Menghapus node pertama dari linked list.
- Fungsi popCenter(): Menghapus node pada posisi tertentu di dalam linked list.
- Fungsi update(): Memperbarui data produk skincare dengan nama produk tertentu.
- Fungsi updateCenter(): Memperbarui data produk skincare pada posisi tertentu di dalam linked list.
- Fungsi deleteAll(): Menghapus semua node dalam linked list.
- Fungsi display(): Menampilkan semua data produk skincare yang ada dalam linked list.

Pada sebuah Fungsi main(): Berisi loop utama yang memberikan opsi kepada pengguna untuk melakukan operasi-operasi seperti menambah data, menghapus data, mengubah data, menampilkan data, dan keluar dari program. Setelah pengguna memilih operasi, program akan menjalankan fungsi yang sesuai dengan pilihan pengguna menggunakan switch-case. Program akan terus berjalan sampai pengguna memilih opsi 8 untuk keluar.

D. Kesimpulan

Single Linked List adalah struktur data di mana setiap simpul hanya memiliki satu pointer yang menunjuk ke simpul berikutnya. Karakteristiknya yaitu efisien dalam penyisipan dan penghapusan elemen di depan atau tengah linked list. Memerlukan lebih sedikit memori karena hanya ada satu pointer per simpul. Tidak memungkinkan navigasi mundur (ke simpul sebelumnya) secara langsung.

Double Linked List adalah struktur data di mana setiap simpul memiliki dua pointer: satu yang menunjuk ke simpul berikutnya (next), dan satu yang menunjuk ke simpul sebelumnya (prev). Memiliki karakteristik memungkinkan navigasi maju dan mundur (ke simpul sebelumnya dan berikutnya). Lebih kompleks dalam implementasi karena ada dua pointer per simpul. Cocok untuk operasi yang memerlukan akses ke simpul sebelumnya (seperti penghapusan dari belakang).

Pilihan antara single linked list dan double linked list tergantung pada kebutuhan aplikasi dan efisiensi yang diinginkan. Single linked list lebih sederhana, sedangkan double linked list lebih fleksibel dalam navigasi.

E. Referensi

Aman sharma.2021.Difference between Singly and Doubly Linked List.
<https://www.prepbytes.com/blog/linked-list/difference-between-a-singly-linked-list-and-a-doubly-linked-list/>

Trivusi.2022.Struktur Data Linked List: Pengertian, Karakteristik, dan Jenis-jenisnya.
https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html#google_vignette