

LAPORAN PRAKTIKUM

MODUL III

SINGLE AND DOUBLE LINKED LIST



Disusun oleh:

Muhammad Dani Ayubi
2311102003
S1 IF-11-A

Dosen Pengampu:

Anggi Zafia, S.T., M.Eng.

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO
2024

BAB I

TUJUAN PRAKTIKUM

1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman

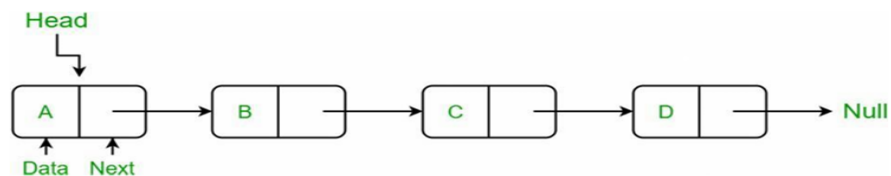
BAB II

DASAR TEORI

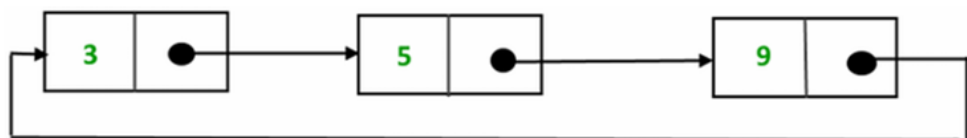
1. Single Linked List

Single linked list adalah struktur data yang terdiri dari serangkaian node yang terhubung satu sama lain melalui pointer. Setiap node dalam linked list berisi dua bagian: data dan pointer ke node selanjutnya. Pointer pada node terakhir menunjuk ke NULL untuk menandakan akhir dari linked list. Keuntungan dari single linked list adalah kemampuannya untuk memudahkan operasi insert dan delete pada elemen di tengah tengah linked list, karena hanya memerlukan pengaturan ulang pointer di beberapa node saja. Namun, linked list ini memiliki keterbatasan dalam melakukan operasi pencarian elemen, karena harus melalui satu per satu node pada linked list. Operasi-operasi yang umum dilakukan pada single linked list adalah:

- Insertion:** menambahkan simpul baru ke dalam linked list pada posisi tertentu.
- Deletion:** menghapus simpul tertentu dari linked list.
- Traversal:** mengunjungi setiap simpul dalam linked list satu per satu.
- Searching:** mencari simpul tertentu dalam linked list.



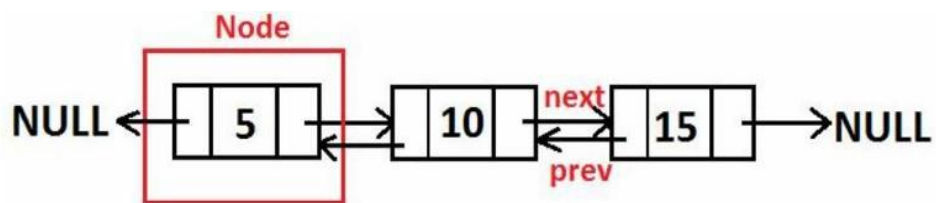
Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya lebih efisien dalam penggunaan memori dibandingkan dengan jenis Linked List lainnya, seperti Double Linked List dan Circular Linked List. Single linked list yang kedua adalah circular linked list. Perbedaan circular linked list dan non circular linked adalah penunjuk next pada node terakhir pada circular linked list akan selalu merujuk ke node pertama



2. Double Linked List

Double linked list adalah struktur data yang serupa dengan single linked list, namun setiap node memiliki dua pointer, yaitu pointer ke node sebelumnya dan pointer ke node selanjutnya. Dengan demikian, double linked list memungkinkan traversal maju-mundur (forward-backward) pada linked list. Keuntungan dari double linked list adalah kemampuan untuk melakukan operasi pencarian dan manipulasi elemen lebih efisien, karena dapat dilakukan traversal maju-mundur pada linked list. Namun, double linked list juga memerlukan penggunaan memori yang lebih besar, karena setiap node menyimpan dua pointer. Selain itu, operasi insert dan delete pada double linked list lebih kompleks, karena memerlukan pengaturan ulang pointer pada node sebelum dan sesudah elemen yang dimanipulasi.

Keuntungan dari Double Linked List adalah memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul dimana saja dengan efisien, sehingga sangat berguna dalam implementasi beberapa algoritma yang membutuhkan operasi tersebut. Selain itu, Double Linked List juga memungkinkan kita untuk melakukan traversal pada list baik dari depan (head) maupun dari belakang (tail) dengan mudah. Namun, kekurangan dari Double Linked List adalah penggunaan memori yang lebih besar dibandingkan dengan Single Linked List, karena setiap simpul membutuhkan satu pointer tambahan. Selain itu, Double Linked List juga membutuhkan waktu eksekusi yang lebih lama dalam operasi penambahan dan penghapusan jika dibandingkan dengan Single Linked List. Representasi sebuah double linked list dapat dilihat pada gambar berikut ini :



Di dalam sebuah linked list, ada 2 pointer yang menjadi penunjuk utama, yakni pointer HEAD yang menunjuk pada node pertama di dalam linked list itu sendiri dan pointer TAIL yang menunjuk pada node paling akhir di dalam linked list. Sebuah linked list dikatakan kosong apabila isi pointer head adalah NULL. Selain itu, nilai pointer prev dari HEAD selalu NULL, karena merupakan data pertama. Begitu pula dengan pointer next dari TAIL yang selalu bernilai NULL sebagai penanda data terakhir.

BAB III

GUIDED

1. Guided 1

Source code

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

/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node
struct Node
{
    // komponen/member
    int data;
    string kata;
    Node *next;
};
Node *head;
Node *tail;
// Inisialisasi Node
void init()
{
    head = NULL;
    tail = NULL;
}
// Pengecekan
bool isEmpty()
{
    if (head == NULL)
        return true;
    else
        return false;
}
// Tambah Depan
void insertDepan(int nilai, string kata)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->kata = kata;
    baru->next = NULL;
    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
}
```

```

    }
    else
    {
        baru->next = head;
        head = baru;
    }
}

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

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

// Tambah Tengah
void insertTengah(int data, string kata, 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;
        baru->kata = kata;
        // 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()
{
    Node *hapus;
    if (isEmpty() == false)
    {
        if (head->next != NULL)
        {
            hapus = head;
            head = head->next;
            delete hapus;
        }
        else
        {
            head = tail = NULL;
        }
    }
    else
    {
        cout << "List kosong!" << endl;
    }
}
// Hapus Belakang
void hapusBelakang()
{
    Node *hapus;

```

```

Node *bantu;
if (isEmpty() == false)
{
    if (head != tail)
    {
        hapus = tail;
        bantu = head;
        while (bantu->next != tail)
        {
            bantu = bantu->next;
        }
        tail = bantu;
        tail->next = NULL;
        delete hapus;
    }
    else
    {
        head = tail = NULL;
    }
}
else
{
    cout << "List kosong!" << endl;
}
}
// Hapus Tengah
void hapusTengah(int posisi)
{
    Node *hapus, *bantu, *bantu2;
    if (posisi < 1 || posisi > hitungList())
    {
        cout << "Posisi di luar jangkauan" << endl;
    }
    else if (posisi == 1)
    {
        cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        int nomor = 1;
        bantu = head;
        while (nomor <= posisi)
        {
            if (nomor == posisi - 1)
            {
                bantu2 = bantu;
            }

```



```

        if (nomor == posisi)
        {
            hapus = bantu;
        }
        bantu = bantu->next;
        nomor++;
    }
    bantu2->next = bantu;
    delete hapus;
}
}
// Ubah Depan
void ubahDepan(int data, string kata)
{
    if (isEmpty() == false)
    {
        head->data = data;
        head->kata = kata;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
// Ubah Tengah
void ubahTengah(int data, string kata, int posisi)
{
    Node *bantu;
    if (isEmpty() == false)
    {
        if (posisi < 1 || posisi > hitungList())
        {
            cout << "Posisi di luar jangkauan" << endl;
        }
        else if (posisi == 1)
        {
            cout << "Posisi bukan posisi tengah" << endl;
        }
        else
        {
            bantu = head;
            int nomor = 1;
            while (nomor < posisi)
            {
                bantu = bantu->next;
                nomor++;
            }

```

```

        bantu->data = data;
        bantu->kata = kata;
    }
}
else
{
    cout << "List masih kosong!" << endl;
}
}

// Ubah Belakang
void ubahBelakang(int data, string kata)
{
    if (isEmpty() == false)
    {
        tail->data = data;
        tail->kata = kata;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}

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

// Tampilkan List
void tampil()
{
    Node *bantu;
    bantu = head;
    if (isEmpty() == false)
    {
        while (bantu != NULL)
        {
            cout << bantu->data << "\t";
            cout << bantu->kata;

```

```

        bantu = bantu->next;
    }
    cout << endl;
}
else
{
    cout << "List masih kosong!" << endl;
}
}
int main()
{
    init();
    insertDepan(3, "satu");
    tampil();
    insertBelakang(5, "dua");
    tampil();
    insertDepan(2, "tiga");
    tampil();
    insertDepan(1, "empat");
    tampil();
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7, "sepuluh" , 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1, "delapan");
    tampil();
    ubahBelakang(8, "sembilan");
    tampil();
    ubahTengah(11, "tujuh", 2);
    tampil();
    return 0;
}

```

Screenshoot program

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\mdani\Vs code> & 'c:\Users\mdani\.vscode\extensions\ms-vscode.cpptools-1.19.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ndjaiifr.txp' '--stdout=Microsoft-MIEngine-Out-cp4ov0x0.g3f' '--stderr=Microsoft-MIEngine-Error-g4hxspt.xwe' '--pid=Microsoft-MIEngine-Pid-jrp1wlh2.xwg' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
3      satu
3      satu5      dua
2      tiga3      satu5      dua
1      empat2      tiga3      satu5      dua
2      tiga3      satu5      dua
2      tiga3      satu
2      tiga7      sepuluh3      satu
2      tiga3      satu
1      delapan3      satu
1      delapan8      sembilan
1      delapan11      tujuh
PS C:\Users\mdani\Vs code>
```

Deskripsi program

Program ini adalah implementasi dari Doubly Linked List (Daftar Taut Ganda) dalam bahasa C++. Doubly Linked List adalah suatu struktur data yang terdiri dari beberapa simpul (node) yang saling terhubung dan setiap simpul memiliki dua pointer yaitu pointer ke simpul sebelumnya (prev) dan pointer ke simpul selanjutnya (next). Pada program ini, terdapat dua kelas yaitu kelas Node dan kelas DoublyLinkedList. Kelas Node merepresentasikan simpul pada Doubly Linked List dan kelas DoublyLinkedList merepresentasikan struktur data Doubly Linked List itu sendiri. Pada int main(), terdapat perulangan while yang akan terus berjalan sampai pengguna memilih pilihan keluar (6). Selama perulangan berjalan, pengguna akan diminta memilih antara menambahkan data, menghapus data, mengubah data, menghapus semua data, atau menampilkan semua data. Pilihan tersebut akan dihandle menggunakan switch case

2. Guided 2

Source code

```
#include <iostream>
using namespace std;
class Node1
{
public:
    int data;
    string kata;
    Node *prev;
    Node *next;
};
class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
    }
    void push(int data, string kata)
    {
        Node *newNode = new Node;
        newNode->data = data;
        newNode->kata = kata;
        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, string oldKata,
string newKata)
    {
        Node *current = head;
        while (current != nullptr)
        {
            if (current->data == oldData)
            {
                current->data = newData;
                current->kata = newKata;
                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->kata;
            current = current->next;
        }
    }

```

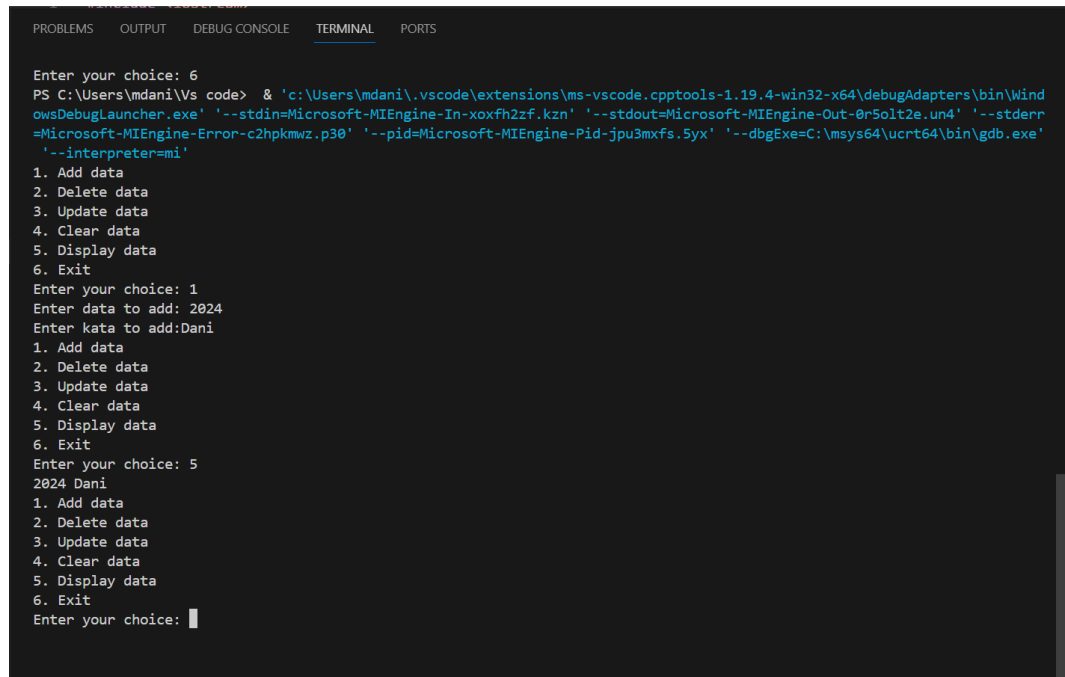
```

        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;
                string kata;
                cout << "Enter data to add: ";
                cin >> data;
                cout << "Enter kata to add:";
                cin >> kata;
                list.push(data, kata);
                break;
            }
            case 2:
            {
                list.pop();
                break;
            }
            case 3:
            {
                int oldData, newData;
                string oldKata, newKata;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                cout << "Enter old kata: ";
                cin >> oldKata;
                cout << "Enter new kata: ";
                cin >> newKata;
            }
        }
    }
}

```

```
        bool updated = list.update(oldData, newData,
oldKata, newKata);
        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



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Enter your choice: 6
PS C:\Users\mdani\Vs code> & 'c:\Users\mdani\.vscode\extensions\ms-vscode.cpptools-1.19.4\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-xoxfh2zf.kzn' '--stdout=Microsoft-MIEngine-Out-0r5olt2e.un4' '--stderr=Microsoft-MIEngine-Error-c2hpkmwz.p30' '--pid=Microsoft-MIEngine-Pid-jpu3mxfs.5yx' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe'
'--interpreter=mi'
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: 2024
Enter kata to add:Dani
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
2024 Dani
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: █
```

Deskripsi program

Program ini adalah implementasi dari Doubly Linked List (Daftar Taut Ganda) dalam bahasa C++. Doubly Linked List adalah suatu struktur data yang terdiri dari beberapa simpul (node) yang saling terhubung dan setiap simpul memiliki dua pointer yaitu pointer ke simpul sebelumnya (prev) dan pointer ke simpul selanjutnya (next). Pada program ini, terdapat dua kelas yaitu kelas Node dan kelas DoublyLinkedList. Kelas Node merepresentasikan simpul pada Doubly Linked List dan kelas DoublyLinkedList merepresentasikan struktur data Doubly Linked List itu sendiri.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    string name;
    int age;
    Node* next;
};

class LinkedList {
private:
    Node* head;

public:
    LinkedList() {
        head = nullptr;
    }

    void insertFirst(string name, int age) { // Using
consistent naming convention
        Node* newNode = new Node;
        newNode->name = name;
        newNode->age = age;
        newNode->next = head;
        head = newNode;
    }

    void insertLast(string name, int age) {
        Node* newNode = new Node;
        newNode->name = name;
        newNode->age = age;
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            return;
        }

        Node* temp = head;
```

```

        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }

    void insertAfter(string name, int age, string nameBefore) {
        Node* newNode = new Node;
        newNode->name = name;
        newNode->age = age;

        Node* temp = head;
        while (temp != nullptr && temp->name != nameBefore) {
            temp = temp->next;
        }

        if (temp == nullptr) {
            cout << "Node with name '" << nameBefore << "' not
found." << endl;
            return;
        }

        newNode->next = temp->next;
        temp->next = newNode;
    }

    void deleteNode(string name) {
        if (head == nullptr) {
            cout << "Linked list is empty." << endl;
            return;
        }

        if (head->name == name) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }

        Node* prev = head;
        Node* temp = head->next;
        while (temp != nullptr && temp->name != name) {
            prev = temp;
            temp = temp->next;
        }

        if (temp == nullptr) {

```

```

        cout << "Node with name '" << name << "' not
found." << endl;
        return;
    }

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

void updateNode(string name, string newName, int newAge) {
    Node* temp = head;
    while (temp != nullptr && temp->name != name) {
        temp = temp->next;
    }

    if (temp == nullptr) {
        cout << "Node with name '" << name << "' not
found." << endl;
        return;
    }

    temp->name = newName;
    temp->age = newAge;
}

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

};

int main() {
    LinkedList myList;

    myList.insertFirst("Muhammmad Dani Ayubi", 20);
    myList.insertFirst("John", 19);
    myList.insertFirst("Jane", 20);
    myList.insertFirst("Michael", 18);
    myList.insertFirst("Yusuke", 19);
    myList.insertFirst("Akechi", 20);
    myList.insertFirst("Hoshino", 18);
    myList.insertFirst("Karin", 18);

    cout << "Data after step (a):" << endl;

```

```

myList.display();
cout << endl;

myList.deleteNode("Akechi");
myList.insertAfter("Futaba", 18, "John");
myList.insertFirst("Igor", 20);
myList.updateNode("Michael", "Reyn", 18);
cout << "Data after all operations:" << endl;
myList.display();

return 0;
}

```

Screenshoot program

```

Data after all operations:
Igor 20
Karin 18
Hoshino 18
Yusuke 19
Reyn 18
Jane 20
John 19
Futaba 18
Muhammad Dani Ayubi 20
PS C:\Users\mdani\Vs code>

```

Deskripsi program

Struktur data "Node" mendefinisikan setiap simpul dalam Linked List, dengan tiga elemen data: nama (string) untuk menyimpan nama seseorang, usia (int) untuk menyimpan usia seseorang, dan "next" (pointer ke Node) yang menunjukkan ke simpul berikutnya dalam Linked List. Kelas "LinkedList" memberikan implementasi operasi dasar untuk Linked List, termasuk "insertAwal" untuk menambahkan simpul baru di awal, "insertAkhir" untuk menambahkan simpul baru di akhir, "insertSetelah" untuk menambahkan simpul baru setelah simpul dengan nama tertentu, "hapus" untuk menghapus simpul dengan nama tertentu, "ubah" untuk mengubah data pada simpul dengan nama tertentu, dan "tampilkan" untuk menampilkan semua data dalam Linked List.

2. Unguided 2

Source code

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

class Node {
public:
    string product_name;
    float price;
    Node* prev;
    Node* next;
};

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

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

    void push(string product_name, float price) {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        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 = temp;
        }
        delete temp;
    }
};
```

```

        tail = nullptr;
    }
    delete temp;
}

bool update(string oldProduct, string newProduct, float
newPrice) {
    Node* current = head;
    while (current != nullptr) {
        if (current->product_name == oldProduct) {
            current->product_name = newProduct;
            current->price = newPrice;
            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->product_name << " (Rp" << current-
>price << ")" << endl;
        current = current->next;
    }
}

void insert(string product_name, float price, int position)
{
    if (position <= 0) {
        push(product_name, price);
        return;
    }
    Node* current = head;

```

```

        for (int i = 1; i < position && current != nullptr;
i++) {
            current = current->next;
        }
        if (current == nullptr) {
            Node* newNode = new Node;
            newNode->product_name = product_name;
            newNode->price = price;
            newNode->prev = tail;
            newNode->next = nullptr;
            if (tail != nullptr) {
                tail->next = newNode;
            } else {
                head = newNode;
            }
            tail = newNode;
        } else {
            Node* newNode = new Node;
            newNode->product_name = product_name;
            newNode->price = price;
            newNode->prev = current->prev;
            newNode->next = current;
            if (current->prev != nullptr) {
                current->prev->next = newNode;
            } else {
                head = newNode;
            }
            current->prev = newNode;
        }
    }

    void remove(int position) {
        if (position <= 0) {
            pop();
            return;
        }
        Node* current = head;
        for (int i = 1; i < position && current != nullptr;
i++) {
            current = current->next;
        }
        if (current == nullptr) {
            return;
        }
        if (current == head) {
            head = current->next;
        } else {

```



```

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

int main() {
    DoublyLinkedList list;
    int choice;
    string productName, newProductName;
    float price, newPrice;
    int position;
    do {
        cout << "1. tambah data\n";
        cout << "2. hapus data\n";
        cout << "3. Update data \n";
        cout << "4. tambah data urutan tertentu insert\n";
        cout << "5. hapus data urutan tertentu remove\n";
        cout << "6. hapus seluruh data\n";
        cout << "7. tampilkan data\n";
        cout << "8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "masukan nama produk: ";
                cin >> productName;
                cout << "masukan harga: ";
                cin >> price;
                list.push(productName, price);
                break;
            case 2:
                list.pop();
                break;
            case 3:
                cout << "masukan nama produk yang lama: ";
                cin >> productName;
                cout << "masukan nama produk yang baru: ";
                cin >> newProductName;
                cout << "masukan harga baru: ";
                cin >> newPrice;

```

```

        if (list.update(productName, newProductName,
newPrice)) {
            cout << "Produk berhasil ditambahkan\n";
        } else {
            cout << "produk tidak ada\n";
        }
        break;
    case 4:
        cout << "masukan nama produk: ";
        cin >> productName;
        cout << "masukan harga: ";
        cin >> price;
        cout << "masukan posisi: ";
        cin >> position;
        list.insert(productName, price, position);
        break;
    case 5:
        cout << "masukan urutan ke-: ";
        cin >> position;
        list.remove(position);
        break;
    case 6:
        list.deleteAll();
        cout << "semua produk berhasil di hapus\n";
        break;
    case 7:
        list.display();
        break;
    case 8:
        cout << "keluar dari progam...\n";
        break;
    default:
        cout << "pilihan salah\n";
        break;
    }
} while (choice != 8);

return 0;
}

```

Screenshoot program

Tambahan Azarine diantara Somethinc dan Skintific

```
1. tambah data
2. hapus data
3. Update data
4. tambah data urutan tertentu insert
5. hapus data urutan tertentu remove
6. hapus seluruh data
7. tampilkan data
8. Exit
Enter your choice: 7
Originote (Rp60000)
Somethinc (Rp150000)
Azarine (Rp65000)
Skintific (Rp100000)
Wardah (Rp50000)
Hanasui (Rp30000)
```

Hapus Wardah

```
1. tambah data
2. hapus data
3. Update data
4. tambah data urutan tertentu insert
5. hapus data urutan tertentu remove
6. hapus seluruh data
7. tampilkan data
8. Exit
Enter your choice: 7
Originote (Rp60000)
Somethinc (Rp150000)
Azarine (Rp65000)
Skintific (Rp100000)
Hanasui (Rp30000)
```

Hanasui Update ke Cleora

```
1. tambah data
2. hapus data
3. Update data
4. tambah data urutan tertentu insert
5. hapus data urutan tertentu remove
6. hapus seluruh data
7. tampilkan data
8. Exit
Enter your choice: 7
Originote (Rp60000)
Somethinc (Rp150000)
Azarine (Rp65000)
Skintific (Rp100000)
Cleora (Rp55000)
```

Deskripsi program

Program ini adalah implementasi dari Doubly Linked List yang memungkinkan pengguna untuk menambah, menghapus, memperbarui, menampilkan, dan menghapus semua data dalam daftar. Setiap node dalam daftar menyimpan informasi tentang nama produk dan harganya. Pada awalnya, head dan tail diatur ke nullptr. Kemudian, program menyediakan menu dengan pilihan untuk melakukan operasi yang berbeda pada daftar. Untuk menambahkan data, pengguna diminta untuk memasukkan nama produk dan harganya, dan data baru ditambahkan ke head daftar. Untuk menghapus data, program akan menghapus node di head daftar. Untuk memperbarui data, pengguna diminta untuk memasukkan nama produk yang ingin diperbarui, dan program akan mencari node dengan nama produk tersebut dan memperbarui informasinya. Untuk menambahkan data pada posisi tertentu, pengguna diminta untuk memasukkan nama produk, harga, dan posisi. Program akan menambahkan data baru pada posisi tertentu dalam daftar. Untuk menghapus data pada posisi tertentu, pengguna diminta untuk memasukkan posisi, dan program akan menghapus data dari daftar pada posisi tersebut. Untuk menghapus semua data dalam daftar, program akan menghapus semua node dalam daftar. Akhirnya, jika pengguna memilih pilihan keluar, program akan berhenti.

BAB IV

KESIMPULAN

materi praktikum single dan double linked list sangatlah penting dalam pemrograman terutama dalam struktur data. Dalam praktikum ini, kita belajar bagaimana membuat dan mengimplementasikan single dan double linked list menggunakan bahasa pemrograman C++. Selain itu, kita juga mempelajari operasi-operasi yang dapat dilakukan pada linked list seperti menambah, menghapus, dan mengupdate data. Materi praktikum ini sangat berguna dalam memecahkan masalah yang memerlukan manipulasi data dengan dinamis. Contohnya, kita dapat menggunakan linked list untuk mengimplementasikan struktur data seperti queue, stack, dan tree. Oleh karena itu, pemahaman tentang single dan double linked list akan sangat membantu dalam pengembangan aplikasi dan pemrograman secara umum. Dalam praktikum single linked list, kita mempelajari cara membuat dan mengimplementasikan linked list dengan satu pointer untuk menunjuk ke node berikutnya. Sedangkan dalam praktikum double linked list, kita mempelajari cara membuat dan mengimplementasikan linked list dengan dua pointer, yaitu satu untuk menunjuk ke node sebelumnya dan satu lagi untuk menunjuk ke node berikutnya. Dengan menggunakan double linked list, kita dapat melakukan beberapa operasi lebih efisien seperti menghapus atau memasukkan node di tengah-tengah linked list. Secara keseluruhan, praktikum single dan double linked list memberikan pemahaman yang sangat penting tentang bagaimana mengelola data secara efisien dengan menggunakan struktur data yang tepat. Hal ini dapat meningkatkan kualitas aplikasi dan membuat kode lebih mudah dipahami, diimplementasikan, dan dikelola.