LAPORAN PRAKTIKUM

MODUL III
SINGLE AND DOUBLE LINKED LIST

Disusun oleh:
Syafanida Khakiki
NIM: 2311102005

Dosen Pengampu:
Muhammad Afrizal Amrustian, S. Kom., M. Kom

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO
2023

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.

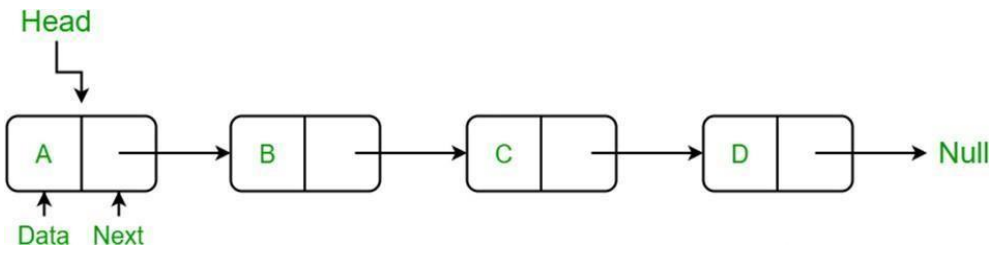
BAB II
DASAR TEORI

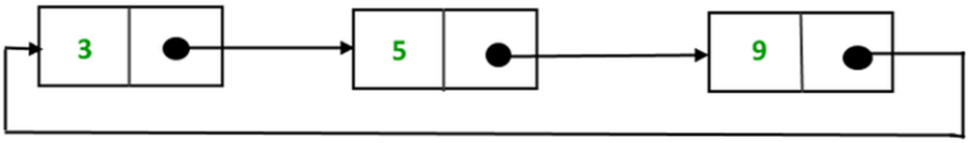
a) Single Linked List

Linked List merupakan suatu bentuk struktur data yang berisi kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung, dinamis, dan terbatas. Setiap elemen dalam linked list dihubungkan ke elemen lain melalui pointer. Masing-masing komponen sering disebut dengan simpul atau node atau verteks. Pointer adalah alamat elemen. Setiap simpul pada dasarnya dibagi atas dua bagian pertama disebut bagian isi atau informasi atau data yang berisi nilai yang disimpan oleh simpul. Bagian kedua disebut bagian pointer yang berisi alamat dari node berikutnya atau sebelumnya. Dengan menggunakan struktur seperti ini, linked list dibentuk dengan cara menunjuk pointer next suatu elemen ke elemen yang mengikutinya. Pointer next pada elemen terakhir merupakan NULL, yang menunjukkan akhir dari suatu list. Elemen pada awal suatu list disebut head dan elemen terakhir dari suatu list disebut tail.

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.



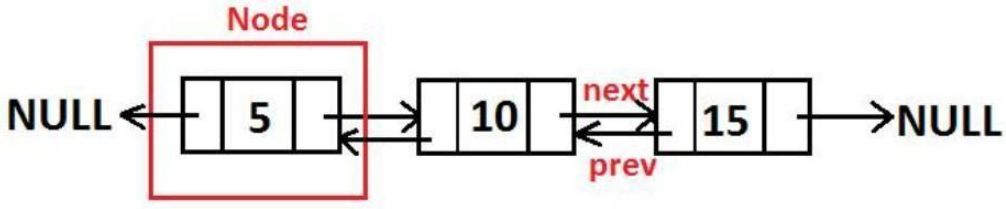


b) Double Linked List

Double Linked List adalah struktur data Linked List yang mirip dengan Single Linked List, namun dengan tambahan satu pointer tambahan pada setiap simpul yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul mana saja secara efisien. Setiap simpul pada Double Linked List memiliki tiga elemen penting, yaitu elemen data (biasanya berupa nilai), pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya.

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 :

```cpp
 #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;
    }hl
    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 << endl;
            cout << bantu->kata << endl;
            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, "lima", 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1, "tujuh");
    tampil();
    ubahBelakang(8, "delapan");
    tampil();
    ubahTengah(11, "sembilan", 2);
    tampil();
    return 0;
}
```

Output :

```
3
satu

3
satu
5
dua

2
tiga
3
satu
5
dua

1
empat
2
tiga
3
satu
5
dua

2
tiga
3
satu
5
dua
```

```
2
tiga
7
lima
3
satu

2
tiga
3
satu

1
tujuh
3
satu

1
tujuh
8
delapan

1
tujuh
11
sembilan
```

Deskripsi program :
Program ini adalah implementasi dari single linked list non sirkular

- **Struktur Data**
  struct Node: Mendefinisikan sebuah node dengan tiga komponen yaitu data (berisi bilangan bulat), kata (berisi string), dan next (pointer ke node berikutnya).

- **Fungsi-fungsi Utama**
  a. **init():** Menginisialisasi list dengan mengatur head dan tail menjadi NULL.
  b. **isEmpty():** Memeriksa apakah list kosong.
  c. **insertDepan():** Menambahkan node baru di depan list.
  d. **insertBelakang():** Menambahkan node baru di belakang list.
  e. **hitungList():** Menghitung jumlah node dalam list.
  f. **insertTengah():** Menambahkan node baru di tengah list pada posisi yang ditentukan.
  g. **hapusDepan():** Menghapus node pertama dari list.
  h. **hapusBelakang():** Menghapus node terakhir dari list.

i. **hapusTengah():** Menghapus node di tengah list pada posisi yang ditentukan.
j. **ubahDepan():** Mengubah data dan kata pada node pertama.
k. **ubahTengah():** Mengubah data dan kata pada node di tengah list pada posisi yang ditentukan.
l. **ubahBelakang():** Mengubah data dan kata pada node terakhir.
m. **clearList():** Menghapus semua node dalam list.
n. **tampil():** Menampilkan isi list.

- **Fungsi main()**
  a. Memulai dengan inisialisasi list menggunakan init().
  b. Menambahkan beberapa node menggunakan insertDepan() dan insertBelakang().
  c. Menampilkan isi list menggunakan tampil().
  d. Melakukan operasi hapus menggunakan hapusDepan(), hapusBelakang(), dan hapusTengah().
  e. Melakukan operasi ubah menggunakan ubahDepan(), ubahBelakang(), dan ubahTengah().
  f. Menampilkan hasil setelah setiap operasi dilakukan.

Program ini menyediakan fungsi-fungsi dasar untuk manipulasi data dalam linked list, seperti penambahan, penghapusan, dan pengubahan data, serta menampilkan isi dari list.

2. Guided 2

Source Code :

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

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

class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = NULL;
        tail = NULL;
```

```
}

void push(int data, string kata)
{
    Node *newNode = new Node;
    newNode->data = data;
    newNode->kata = kata;
    newNode->prev = NULL;
    newNode->next = head;
    if (head != NULL)
    {
        head->prev = newNode;
    }
    else
    {
        tail = newNode;
    }
    head = newNode;
}

void pop()
{
    if (head == NULL)
    {
        return;
    }
    Node *temp = head;
    head = head->next;
    if (head != NULL)
    {
        head->prev = NULL;
    }
    else
    {
        tail = NULL;
    }
    delete temp;
}

bool update(int oldData, int newData, string newKata)
{
    Node *current = head;
    while (current != NULL)
    {
        if (current->data == oldData)
```

```
            {
                current->data = newData;
                current->kata = newKata;
                return true;
            }
            current = current->next;
        }
        return false;
    }

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

    void display()
    {
        Node *current = head;
        while (current != NULL)
        {
            cout << current->data << " ";
            cout << current->kata << endl;
            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 newKata;
            cout << "Enter old data: ";
            cin >> oldData;
            cout << "Enter new data: ";
            cin >> newData;
            cout << "Enter new kata: ";
            cin >> newKata;
            bool updated = list.update(oldData,
                                       newData, 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;
}
```

Output :

```
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter old data: 12
Enter new data: 24
Enter new kata: AKU
Data not found
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice:
```

Deskripsi Program :

Program ini merupakan implementasi dari Double Linked List dalam bahasa pemrograman C++. Struktur data Double Linked List ini terdiri dari simpul-simpul yang memiliki dua pointer yaitu prev yang menunjuk ke simpul sebelumnya dan next yang menunjuk ke simpul selanjutnya.

- **Class dan Metode**

  Class Node:

a. int data: Menyimpan data bertipe integer.
b. string kata: Menyimpan data bertipe string.
c. Node *prev: Pointer yang menunjuk ke simpul sebelumnya.
d. Node *next: Pointer yang menunjuk ke simpul selanjutnya.

- **Kelas DoublyLinkedList:**

a. Node *head: Pointer ke simpul pertama dalam daftar.
b. Node *tail: Pointer ke simpul terakhir dalam daftar.
c. push(int data, string kata): Menambahkan simpul baru di awal daftar.
d. pop(): Menghapus simpul pertama dari daftar.
e. update(int oldData, int newData, string newKata): Mengupdate data dan kata pada simpul dengan data tertentu.
f. deleteAll(): Menghapus semua simpul dari daftar.
g. display(): Menampilkan isi dari daftar.

- **Fungsi main()**

a. Memulai dengan membuat objek dari kelas DoublyLinkedList.
b. Program memberikan beberapa opsi kepada pengguna:
- Menambahkan data baru ke dalam daftar.
- Menghapus data dari daftar.
- Mengupdate data dalam daftar.
- Menghapus semua data dari daftar.
- Menampilkan isi dari daftar.
- Keluar dari program.

c. Setelah pengguna memilih suatu opsi, program akan menjalankan tindakan yang sesuai dengan pilihan tersebut.
d. Program akan terus berjalan hingga pengguna memilih untuk keluar.

BAB IV
UNGUIDED

1. Unguided 1

Source code :

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

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

class LinkedList {
private:
    Mahasiswa* head;

public:
    LinkedList() {
        head = NULL;
    }

    void tambahMahasiswa(string nama, int usia) {
        Mahasiswa* newMahasiswa = new Mahasiswa;
        newMahasiswa->nama = nama;
        newMahasiswa->usia = usia;
        newMahasiswa->next = head;
        head = newMahasiswa;
    }

    void hapusMahasiswa(string nama) {
        Mahasiswa* current = head;
        Mahasiswa* prev = NULL;

        while (current != NULL) {
            if (current->nama == nama) {
                if (prev == NULL) {
                    head = current->next;
```

```
                } else {
                    prev->next = current->next;
                }
                delete current;
                cout << "Data mahasiswa dengan nama " << nama << " telah
dihapus." << endl;
                return;
            }
            prev = current;
            current = current->next;
        }

        cout << "Data mahasiswa dengan nama " << nama << " tidak ditemukan."
<< endl;
    }

    void tambahMahasiswaDiUrutan(string nama, int usia, int urutan) {
        Mahasiswa* newMahasiswa = new Mahasiswa;
        newMahasiswa->nama = nama;
        newMahasiswa->usia = usia;

        if (urutan == 1 || head == NULL) {
            newMahasiswa->next = head;
            head = newMahasiswa;
        } else {
            Mahasiswa* current = head;
            int posisi = 1;

            while (posisi < urutan - 1 && current->next != NULL) {
                current = current->next;
                posisi++;
            }

            newMahasiswa->next = current->next;
            current->next = newMahasiswa;
        }
    }

    void gantiDataMahasiswa(string nama, string namaBaru, int usiaBaru) {
        Mahasiswa* current = head;

        while (current != NULL) {
            if (current->nama == nama) {
                current->nama = namaBaru;
                current->usia = usiaBaru;
```

```
                cout << "Data mahasiswa dengan nama " << nama << " telah
diubah." << endl;
                return;
            }
            current = current->next;
        }

        cout << "Data mahasiswa dengan nama " << nama << " tidak ditemukan."
<< endl;
    }

    void tampilkanDataAwal() {
        Mahasiswa* current = head;

        if (current == NULL) {
            cout << "Linked list kosong." << endl;
            return;
        }

        cout << "Data Awal Mahasiswa:" << endl;
        while (current != NULL) {
            cout << current->nama << " " << current->usia << endl;
            current = current->next;
        }
        cout << endl;
    }

    void tampilkanMahasiswa() {
        Mahasiswa* current = head;

        if (current == NULL) {
            cout << "Linked list kosong." << endl;
            return;
        }

        cout << "Data Mahasiswa:" << endl;
        cout << "-----------------" << endl;
        while (current != NULL) {
            cout << current->nama << " " << current->usia << endl;
            current = current->next;
        }
        cout << endl;
    }
};
```

```cpp
int main() {
    LinkedList daftarMahasiswa;

    daftarMahasiswa.tambahMahasiswa("Karin", 18);
    daftarMahasiswa.tambahMahasiswa("Hoshino", 18);
    daftarMahasiswa.tambahMahasiswa("Akechi", 20);
    daftarMahasiswa.tambahMahasiswa("Yusuke", 19);
    daftarMahasiswa.tambahMahasiswa("Michael", 18);
    daftarMahasiswa.tambahMahasiswa("Jane", 20);
    daftarMahasiswa.tambahMahasiswa("John", 19);

    daftarMahasiswa.tampilkanDataAwal();

    int pilihan;
    string nama, namaBaru;
    int usia, usiaBaru, urutan;

    do {
        cout << "\nMenu:\n";
        cout << "1. Tambah Data Mahasiswa\n";
        cout << "2. Hapus Data Mahasiswa\n";
        cout << "3. Tambah Data Mahasiswa di Urutan Tertentu\n";
        cout << "4. Ubah Data Mahasiswa\n";
        cout << "5. Tampilkan Semua Data Mahasiswa\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;

        switch (pilihan) {
            case 1:
                cout << "Masukkan nama mahasiswa: ";
                cin >> nama;
                cout << "Masukkan usia mahasiswa: ";
                cin >> usia;
                daftarMahasiswa.tambahMahasiswa(nama, usia);
                break;
            case 2:
                cout << "Masukkan nama mahasiswa yang ingin dihapus: ";
                cin >> nama;
                daftarMahasiswa.hapusMahasiswa(nama);
                break;
            case 3:
                cout << "Masukkan nama mahasiswa: ";
                cin >> nama;
                cout << "Masukkan usia mahasiswa: ";
```

```
                cin >> usia;
                cout << "Masukkan urutan: ";
                cin >> urutan;
                daftarMahasiswa.tambahMahasiswaDiUrutan(nama, usia, urutan);
                break;
            case 4:
                cout << "Masukkan nama mahasiswa yang ingin diubah: ";
                cin >> nama;
                cout << "Masukkan nama baru: ";
                cin >> namaBaru;
                cout << "Masukkan usia baru: ";
                cin >> usiaBaru;
                daftarMahasiswa.gantiDataMahasiswa(nama, namaBaru, usiaBaru);
                break;
            case 5:
                daftarMahasiswa.tampilkanMahasiswa();
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                break;
            default:
                cout << "Pilihan tidak valid. Silakan pilih lagi." << endl;
        }
    } while (pilihan != 6);

 return 0;
}
```

Output :

```
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
Pilih menu: 3
Masukkan Nama Produk yang Ingin Diupdate: Hanasui
Masukkan Nama Produk Baru: Cleora
Masukkan Harga Baru: 55000

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
Pilih menu: 7
Data Produk:
Nama Produk     Harga
======================
Originote       60000
Somethinc       150000
Azarine         65000
Skintific       100000
Cleora          55000
```

```
Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Tambah Data Mahasiswa di Urutan Tertentu
4. Ubah Data Mahasiswa
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilih menu: 4
Masukkan nama mahasiswa yang ingin diubah: Michael
Masukkan nama baru: Reyn
Masukkan usia baru: 18
Data mahasiswa dengan nama Michael telah diubah.

Menu:
1. Tambah Data Mahasiswa
2. Hapus Data Mahasiswa
3. Tambah Data Mahasiswa di Urutan Tertentu
4. Ubah Data Mahasiswa
5. Tampilkan Semua Data Mahasiswa
6. Keluar
Pilih menu: 5
Data Mahasiswa:
----------------
Igor 20
Syafanida 20
John 19
Futaba 18
Jane 20
Reyn 18
Yusuke 19
Hoshino 18
Karin 18
```

```
Menu:
1. Tambah Data
2. Hapus Data
3. Tambah Data
4. Ubah Data
5. Tampilkan
6. Keluar
Pilih menu: 3
Masukkan nama
Masukkan usia
Masukkan uruta

Menu:
1. Tambah Data
2. Hapus Data
3. Tambah Data
4. Ubah Data
5. Tampilkan
6. Keluar
Pilih menu: 1
Masukkan nama
Igor
Masukkan usia
```

Deskripsi Program :

Struktur Data

1. **Struktur Mahasiswa:**

- **string nama:** Menyimpan nama mahasiswa.
- **int usia:** Menyimpan usia mahasiswa.
- **Mahasiswa* next**: Pointer yang menunjuk ke mahasiswa selanjutnya dalam linked list.

2. **Kelas LinkedList:**

- **Mahasiswa* head:** Pointer yang menunjuk ke mahasiswa pertama dalam linked list.

Metode dalam Kelas LinkedList

1. tambahMahasiswa(string nama, int usia): Menambahkan data mahasiswa baru ke dalam linked list.
2. hapusMahasiswa(string nama): Menghapus data mahasiswa dari linked list berdasarkan nama.
3. tambahMahasiswaDiUrutan(string nama, int usia, int urutan): Menambahkan data mahasiswa pada urutan tertentu dalam linked list.
4. gantiDataMahasiswa(string nama, string namaBaru, int usiaBaru): Mengubah data mahasiswa berdasarkan nama.
5. tampilkanDataAwal(): Menampilkan seluruh data mahasiswa yang tersimpan di awal linked list.
6. tampilkanMahasiswa(): Menampilkan seluruh data mahasiswa yang tersimpan dalam linked list.

Fungsi main()

- Membuat objek dari kelas LinkedList.
- Menambahkan beberapa data mahasiswa ke dalam linked list secara statis.
- Menampilkan seluruh data mahasiswa yang tersimpan di awal linked list.
- Memberikan pilihan menu kepada pengguna untuk melakukan operasi seperti menambah, menghapus, atau mengubah data mahasiswa, serta menampilkan seluruh data mahasiswa atau keluar dari program.
- Program akan terus berjalan hingga pengguna memilih untuk keluar.

2. **Unguided 2**

Source Code :

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

struct Product {
    string nama_produk;
    int harga;
```

```
    Product* prev;
    Product* next;
};

class DoubleLinkedList {
private:
    Product* head;
    Product* tail;

public:
    DoubleLinkedList() {
        head = NULL;
        tail = NULL;
    }

    void addProduct(string nama_produk, int harga) {
        Product* newProduct = new Product;
        newProduct->nama_produk = nama_produk;
        newProduct->harga = harga;
        newProduct->prev = NULL;
        newProduct->next = NULL;

        if (head == NULL) {
            head = newProduct;
            tail = newProduct;
        } else {
            tail->next = newProduct;
            newProduct->prev = tail;
            tail = newProduct;
        }
    }

    void removeProduct(string nama_produk) {
        Product* current = head;
        while (current != NULL) {
            if (current->nama_produk == nama_produk) {
                if (current == head) {
                    head = head->next;
                    if (head != NULL) {
                        head->prev = NULL;
                    }
                } else if (current == tail) {
                    tail = tail->prev;
                    tail->next = NULL;
                } else {
```

```
                current->prev->next = current->next;
                current->next->prev = current->prev;
            }
            delete current;
            return;
        }
        current = current->next;
    }
    cout << "Produk tidak ditemukan." << endl;
}

void updateProduct(string old_nama_produk, string new_nama_produk, int
new_harga) {
    Product* current = head;
    while (current != NULL) {
        if (current->nama_produk == old_nama_produk) {
            current->nama_produk = new_nama_produk;
            current->harga = new_harga;
            return;
        }
        current = current->next;
    }
    cout << "Produk tidak ditemukan." << endl;
}

void displayProducts() {
    cout << "Nama Produk\tHarga" << endl;
    cout << "======================" << endl;
    Product* current = head;
    while (current != NULL) {
        cout << setw(12) << left << current->nama_produk << "\t" <<
current->harga << endl;
        current = current->next;
    }
    cout << endl;
}

void addProductBetween(string prevProductName, string newProductName, int
newPrice) {
    Product* current = head;
    while (current != NULL) {
        if (current->nama_produk == prevProductName) {
            Product* newProduct = new Product;
            newProduct->nama_produk = newProductName;
            newProduct->harga = newPrice;
```

```
            newProduct->prev = current;
            newProduct->next = current->next;
            if (current->next != NULL) {
                current->next->prev = newProduct;
            }
            current->next = newProduct;
            return;
        }
        current = current->next;
    }
    cout << "Produk sebelumnya tidak ditemukan." << endl;
}

void removeAtPosition(int position) {
    if (head == NULL) {
        cout << "List kosong." << endl;
        return;
    }

    Product* current = head;
    int count = 1;

    while (current != NULL && count != position) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        cout << "Posisi tidak valid." << endl;
        return;
    }

    if (current == head) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        delete current;
    } else if (current == tail) {
        tail = tail->prev;
        tail->next = NULL;
        delete current;
    } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
```

```
            delete current;
        }
    }

    void removeAll() {
        Product* current = head;
        while (current != NULL) {
            Product* temp = current;
            current = current->next;
            delete temp;
        }
        head = NULL;
        tail = NULL;
    }
};

int main() {
    DoubleLinkedList productList;

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

    int choice;
    string prevProductName, newProductName;
    int newPrice;

    do {
        cout << "\nToko 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 << "Pilih menu: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Masukkan Nama Produk: ";
```

```
                cin >> newProductName;
                cout << "Masukkan Harga: ";
                cin >> newPrice;
                productList.addProduct(newProductName, newPrice);
                break;
            case 2:
                cout << "Masukkan Nama Produk yang Ingin Dihapus: ";
                cin >> newProductName;
                productList.removeProduct(newProductName);
                break;
            case 3:
                cout << "Masukkan Nama Produk yang Ingin Diupdate: ";
                cin >> prevProductName;
                cout << "Masukkan Nama Produk Baru: ";
                cin >> newProductName;
                cout << "Masukkan Harga Baru: ";
                cin >> newPrice;
                productList.updateProduct(prevProductName, newProductName,
newPrice);
                break;
            case 4:
                cout << "Masukkan Nama Produk Sebelumnya: ";
                cin >> prevProductName;
                cout << "Masukkan Nama Produk Baru: ";
                cin >> newProductName;
                cout << "Masukkan Harga Baru: ";
                cin >> newPrice;
                productList.addProductBetween(prevProductName, newProductName,
newPrice);
                break;
            case 5:
            {
                string productName;
                cout << "Masukkan Nama Produk yang Ingin Dihapus: ";
                cin >> productName;
                productList.removeProduct(productName);
                break;
            }
            case 6:
                productList.removeAll();
                cout << "Semua data telah dihapus." << endl;
                break;
            case 7:
                cout << "Data Produk:" << endl;
                productList.displayProducts();
```

```
                break;
            case 8:
                cout << "Terima kasih!" << endl;
                break;
            default:
                cout << "Pilihan tidak valid. Silakan pilih lagi." << endl;
        }
    } while (choice != 8);

 return ;
}
```

Output :

```
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
Pilih menu: 4
Masukkan Nama Produk Sebelumnya: Somethinc
Masukkan Nama Produk Baru: Azarine
Masukkan Harga Baru: 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
Pilih menu: 2
Masukkan Nama Produk yang Ingin 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
Pilih menu: 3
Masukkan Nama Produk yang Ingin Diupdate: Hanasui
Masukkan Nama Produk Baru: Cleora
Masukkan Harga Baru: 55000

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
Pilih menu: 7
Data Produk:
Nama Produk     Harga
======================
Originote       60000
Somethinc       150000
Azarine         65000
Skintific       100000
Cleora          55000
```

Deskripsi Program :

Program ini merupakan aplikasi sederhana untuk mengelola daftar produk menggunakan ldouble linked list. Program ini menyediakan beberapa fitur seperti menambahkan produk, menghapus produk, mengupdate data produk, menambahkan produk pada urutan tertentu, menghapus produk pada urutan tertentu, menghapus seluruh data produk, dan menampilkan seluruh data produk yang tersimpan.

Struktur Data

1. **Struktur Product:**
   - string nama_produk: Menyimpan nama produk.
   - int harga: Menyimpan harga produk.
   - Product* prev: Pointer yang menunjuk ke produk sebelumnya dalam linked list.
   - Product* next: Pointer yang menunjuk ke produk selanjutnya dalam linked list.

2. **Kelas DoubleLinkedList:**
   - Product* head: Pointer yang menunjuk ke produk pertama dalam linked list.
   - Product* tail: Pointer yang menunjuk ke produk terakhir dalam linked list.

Metode dalam Kelas DoubleLinkedList

1. addProduct(string nama_produk, int harga): Menambahkan produk baru ke dalam linked list.
2. removeProduct(string nama_produk): Menghapus produk dari linked list berdasarkan nama.
3. updateProduct(string old_nama_produk, string new_nama_produk, int new_harga): Mengupdate data produk berdasarkan nama.
4. displayProducts(): Menampilkan seluruh data produk yang tersimpan dalam linked list.
5. addProductBetween(string prevProductName, string newProductName, int newPrice): Menambahkan produk pada urutan tertentu dalam linked list.
6. removeAtPosition(int position): Menghapus produk pada posisi tertentu dalam linked list.
7. removeAll(): Menghapus seluruh data produk dari linked list.

Fungsi main()

- Membuat objek dari kelas DoubleLinkedList.
- Menambahkan beberapa data produk ke dalam linked list secara statis.
- Memberikan pilihan menu kepada pengguna untuk melakukan operasi seperti menambah, menghapus, atau mengubah data produk, menambahkan produk pada urutan tertentu, menghapus produk pada urutan tertentu, menghapus seluruh data produk, atau keluar dari program.

BAB IV

KESIMPULAN

1. Single Linked List: Merupakan struktur data yang terdiri dari kumpulan simpul yang saling terhubung satu sama lain dengan menggunakan pointer next. Setiap

simpul memiliki dua bagian utama, yaitu data dan pointer next yang menunjuk ke simpul berikutnya. Operasi umum pada Single Linked List meliputi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Single Linked List lebih efisien dalam penggunaan memori karena hanya membutuhkan satu pointer untuk setiap simpulnya.

2. Circular Linked List: Jenis khusus dari Single Linked List di mana pointer next pada node terakhir akan selalu merujuk kembali ke node pertama, membentuk suatu lingkaran. Circular Linked List memungkinkan traversal dari node pertama ke node terakhir dengan lebih efisien tanpa harus mengubah arah pointer. Ini dapat membantu dalam beberapa implementasi algoritma.

3. Double Linked List: Merupakan struktur data Linked List yang mirip dengan Single Linked List, namun setiap simpul memiliki tambahan satu pointer tambahan yaitu pointer prev yang menunjuk ke simpul sebelumnya. Double Linked List memungkinkan untuk melakukan operasi penghapusan dan penambahan pada simpul mana saja dengan efisien. Keuntungan utamanya adalah memungkinkan traversal dari depan (head) maupun dari belakang (tail) dengan mudah, serta operasi penghapusan dan penambahan yang efisien. Namun, penggunaan memori lebih besar dan waktu eksekusi lebih lama dibandingkan dengan Single Linked List.

Dalam prakteknya, pemilihan jenis Linked List yang tepat bergantung pada kebutuhan dan karakteristik dari aplikasi atau algoritma yang akan diimplementasikan. Misalnya, untuk aplikasi yang memerlukan operasi traversal dari belakang ke depan atau sebaliknya, Double Linked List lebih cocok digunakan. Sedangkan, jika memori terbatas atau operasi penambahan dan penghapusan sering terjadi di ujung linked list, Single Linked List dapat menjadi pilihan yang lebih efisien.

DAFTAR PUSTAKA

Asisten Praktikum (2024). Array. Learning Management System