

**LAPORAN PRAKTIKUM STRUKTUR DATA DAN
PEMROGRAMAN**

**MODUL 3
SINGLE AND DOUBLE LINKED LIST**



Disusun oleh :

ZEFANYA.B.T.TARIGAN

2311102028

IF-11-A

Dosen Pengampu :

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

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

BAB I

TUJUAN PRAKTIKUM

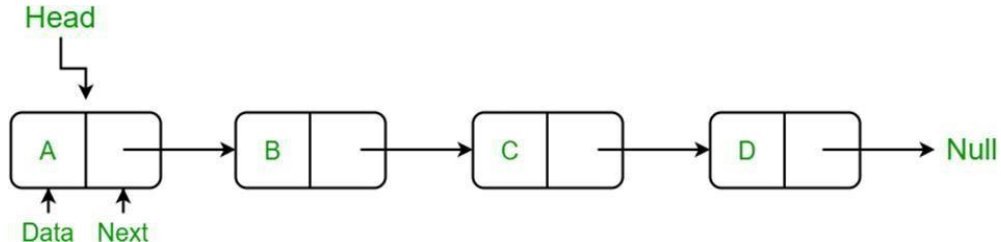
1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman

BAB II

DASAR TEORI

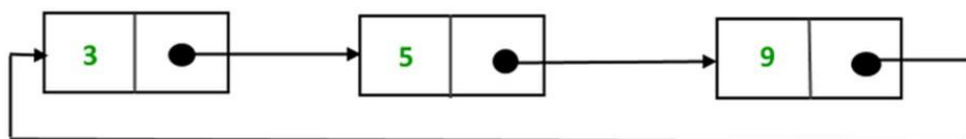
a) Single Linked List

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.

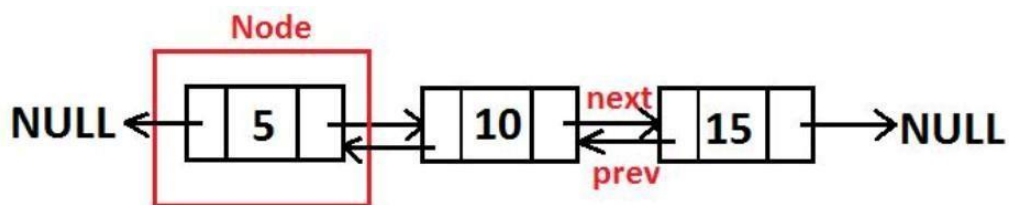


b) Double Linked List

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

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)

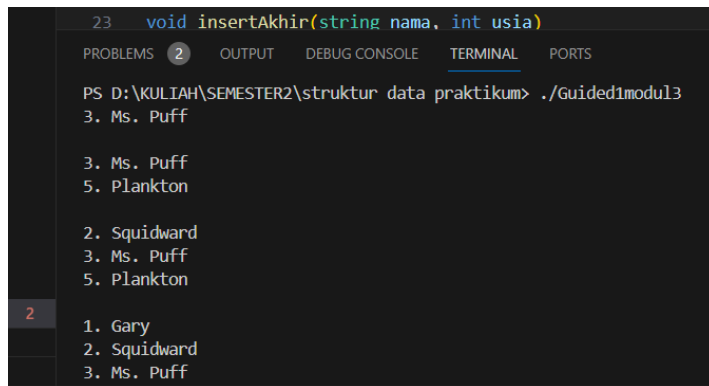
```



```
{
    while (bantu != NULL)
    {
        cout << bantu->data << ". ";
        cout << bantu->kata << endl;
        bantu = bantu->next;
    }
    cout << endl;
}
else
{
    cout << "List masih kosong!" << endl;
}
}

int main()
{
    init();
    insertDepan(3, "Ms. Puff");
    tampil();
    insertBelakang(5, "Plankton");
    tampil();
    insertDepan(2, "Squidward");
    tampil();
    insertDepan(1, "Gary");
    tampil();
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7, "Spongebob", 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1, "Magor");
    tampil();
    ubahBelakang(8, "Yaju");
    tampil();
    ubahTengah(11, "Suki", 2);
    tampil();
    return 0;
}
```

SCREENSHOOT PROGRAM



```
23 void insertAkhir(string nama, int usia)
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\KULIAH\SEMESTER2\struktur data praktikum> ./Guided1modul3
3. Ms. Puff

3. Ms. Puff
5. Plankton

2. Squidward
3. Ms. Puff
5. Plankton

2
1. Gary
2. Squidward
3. Ms. Puff
```

DESKRIPSI PROGRAM

Program ini mendemonstrasikan berbagai operasi pada single linked list, seperti:

Menambahkan node di depan dan belakang list.

Menghitung jumlah node dalam list.

Menambahkan node di posisi tertentu dalam list.

Menghapus node di depan, belakang, dan posisi tertentu dalam list.

Mengubah data dan kata pada node di depan, belakang, dan posisi tertentu dalam list.

Menghapus semua node dalam list.

Menampilkan data dan kata dari semua node dalam

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 = 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 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 << " ";  
  
    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;

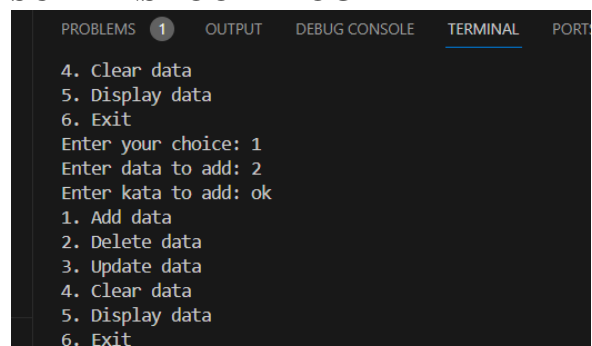
}

}

return 0;

}
```

SCREENSHOOT PROGRAM



DESKRIPSI PROGRAM

Program ini mendemonstrasikan struktur data doubly linked list dalam C++. Ini memungkinkan untuk menjelajah ke depan dan ke belakang melalui daftar. Berikut adalah deskripsi komponen utamanya:

1. Kelas Node:

Mewakili sebuah node dalam daftar.

Berisi atribut:

data: Integer untuk menyimpan data.

kata: String untuk menyimpan string.

prev: Pointer ke node sebelumnya.

next: Pointer ke node berikutnya.

2. Kelas DoublyLinkedList:

Mengelola struktur linked list.

Berisi metode utama:

-push(data, kata): Menambahkan node baru di depan daftar.

-pop(): Menghapus node di depan daftar.

-update(oldData, newData, newKata): Mengubah data dan kata pada node dengan nilai oldData.

-deleteAll(): Menghapus semua node dalam daftar.

-display(): Menampilkan data dan kata dari semua node dalam daftar.

3. Fungsi main():

-Menciptakan objek DoublyLinkedList.

-Menyajikan menu untuk pengguna agar dapat berinteraksi dengan daftar:

1.Menambahkan data.

2.Menghapus data.

3.Memperbarui data.

4.Menghapus semua data.

5.Menampilkan data.

Keluar dari program.

Program ini menunjukkan cara membuat dan memanipulasi doubly linked list dalam C++, memberikan fondasi untuk membangun struktur data yang lebih kompleks.

BAB IV

UNGUIDED

UNGUIDED1

SOURCE CODE

```
#include <iostream>

//Zefanya-Brana-Tertius-Tarigan/2311102028/IF-11-A

using namespace std;

class Node
{
public:
    string name;
    int age;
    Node *next;
};

class LinkedList
{
public:
    Node *head;
    LinkedList()
    {
        head = NULL;
    }
    void insertAtFront(string name, int age)
    {
        Node *newNode = new Node();
        newNode->name = name;
        newNode->age = age;
        newNode->next = head;
```

```
head = newNode;

}

void insertAtEnd(string name, int age)
{
    Node *newNode = new Node();
    newNode->name = name;
    newNode->age = age;
    newNode->next = NULL;
    if (head == NULL)
    {
        head = newNode;
        return;
    }
    Node *temp = head;
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = newNode;
}

void insertAfter(string name, int age, string keyName)
{
    Node *temp = head;
    while (temp != NULL)
    {
        if (temp->name == keyName)
        {
            Node *newNode = new Node();
            newNode->name = name;
            newNode->age = age;
            newNode->next = temp->next;
            temp->next = newNode;
            return;
        }
    }
```

```

temp = temp->next;
}
cout << keyName << " not found in the list." << endl;
}

void updateNode(string name, int age)
{
Node *temp = head;
while (temp != NULL)
{
if (temp->name == name)
{
temp->age = age;
return;
}
temp = temp->next;
}
cout << name << " not found in the list." << endl;
}

void deleteNode(string name)
{
Node *temp = head;
Node *prev = NULL;
while (temp != NULL)
{
if (temp->name == name)
{
if (prev == NULL)
{
head = temp->next;
}
else
{
prev->next = temp->next;

```

```

}

delete temp;

return;

}

prev = temp;
temp = temp->next;

}

cout << name << " not found in the list." << endl;

}

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

// Display all nodes
void display()
{
Node *temp = head;
while (temp != NULL)
{
cout << "Name: " << temp->name << ", Age: " <<
temp->age << endl;
temp = temp->next;
}
}

};

// Main function

```

```
int main()
{
    LinkedList list;
    int choice;
    string name, keyName;
    int age;
    do
    {
        cout << endl;
        cout << "MENU" << endl;
        cout << "1. Add data" << endl;
        cout << "2. Update data" << endl;
        cout << "3. Delete data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice)
        {
            case 1:
                cout << "Enter name: ";
                cin >> name;
                cout << "Enter age: ";
                cin >> age;
                //Output
                //a. Masukkan data sesuai urutan
                list.insertAtFront(name, age);
                break;
            case 2:
                cout << "Enter name to update: ";
                cin >> name;
                cout << "Enter new age: ";
                cin >> age;
```

```
list.updateNode(name, age);
break;
case 3:
cout << "Enter name to delete: ";
cin >> name;
list.deleteNode(name);
break;
case 4:
list.clearAll();
break;
case 5:
list.display();
break;
case 6:
cout << "Exiting program..." << endl;
break;
default:
cout << "Invalid choice." << endl;
}
} while (choice != 6);
return 0;
}
```

Output a

```
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Karin, Age: 18
Name: Hoshino, Age: 18
Name: Akechi, Age: 20
Name: Yusuke, Age: 19
Name: Michael, Age: 18
Name: Jane, Age: 20
Name: Jhon, Age: 19
```

Output b

```
5. Display data
6. Exit
Enter your choice: 5
Name: Karin, Age: 18
Name: Hoshino, Age: 18
Name: Yusuke, Age: 19
Name: Michael, Age: 18
Name: Jane, Age: 20
Name: Jhon, Age: 19
```

Output c

```
5. Display data
6. Exit
Enter your choice: 5
Name: Futaba, Age: 18
Name: Karin, Age: 18
Name: Hoshino, Age: 18
Name: Yusuke, Age: 19
Name: Michael, Age: 18
Name: Jane, Age: 20
Name: Jhon, Age: 19
```

Output d

```
6. Exit
Enter your choice: 5
Name: Igor, Age: 20
Name: Futaba, Age: 18
Name: Karin, Age: 18
Name: Hoshino, Age: 18
Name: Yusuke, Age: 19
Name: Michael, Age: 18
Name: Jane, Age: 20
Name: Jhon, Age: 19
```


Output e dan f

```
6. Exit
Enter your choice: 5
Name: Jhon, Age: 19
Name: Jane, Age: 20
Name: Reyn, Age: 18
Name: Yusuke, Age: 19
Name: Hoshino, Age: 18
Name: Karin, Age: 18
Name: Futaba, Age: 18
Name: Igor, Age: 20
```

Deskripsi Program

Menciptakan objek LinkedList bernama list.

Menampilkan menu secara berulang kepada pengguna hingga pengguna memilih opsi keluar (6).

Menu tersebut berisi pilihan untuk:

Menambah data (Pilihan 1).

Memperbarui data (Pilihan 2).

Menghapus data (Pilihan 3).

Menghapus semua data (Pilihan 4).

Menampilkan semua data (Pilihan 5).

Keluar dari program (Pilihan 6).

Berdasarkan pilihan pengguna, fungsi terkait dari kelas LinkedList akan dipanggil.

Unguided 2

Source Code

```
#include <iostream>
#include <string>
using namespace std;
struct Node
{
    string nama;
    int harga;
    Node *prev;
    Node *next;
};
class DoubleLinkedList
{
private:
    Node *head;
    Node *tail;
    int size;
public:
    DoubleLinkedList()
    {
        head = NULL;
        tail = NULL;
        size = 0;
    }
    void addData(string nama, int harga)
    {
        Node *node = new Node;
        node->nama = nama;
        node->harga = harga;
        node->prev = tail;
        node->next = NULL;
```

```

if (head == NULL)
{
    head = node;
    tail = node;
}
else
{
    tail->next = node;
    tail = node;
}
size++;
}

void addDataAt(int index, string nama, int harga)
{
    if (index < 0 || index > size)
    {
        cout << "Index out of bounds" << endl;
        return;
    }
    Node *node = new Node;
    node->nama = nama;
    node->harga = harga;
    if (index == 0)
    {
        node->prev = NULL;
        node->next = head;
        head->prev = node;
        head = node;
    }
    else if (index == size)
    {
        node->prev = tail;
        node->next = NULL;
        tail->next = node;
    }
}

```

```

tail = node;

}

else
{
Node *current = head;
for (int i = 0; i < index - 1; i++)
{
current = current->next;
}
node->prev = current;
node->next = current->next;
current->next->prev = node;
current->next = node;
}
size++;
}

void deleteDataAt(int index)
{
if (index < 0 || index >= size)
{
cout << "Index out of bounds" << endl;
return;
}

if (index == 0)
{
Node *temp = head;
head = head->next;
head->prev = NULL;
delete temp;
}

else if (index == size - 1)
{
Node *temp = tail;
tail = tail->prev;
}
}

```

```

tail->next = NULL;

delete temp;

}

else
{
Node *current = head;
for (int i = 0; i < index; i++)
{
current = current->next;
}

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

size--;
}

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

tail = NULL;
size = 0;
}

void displayData()
{
cout << "Nama Produk\tHarga" << endl;
Node *current = head;
while (current != NULL)
{

```

```

tail->next = NULL;

delete temp;

}

else
{
Node *current = head;
for (int i = 0; i < index; i++)
{
current = current->next;
}
current->prev->next = current->next;
current->next->prev = current->prev;
delete current;
}

size--;
}

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

tail = NULL;

size = 0;
}

void displayData()
{
cout << "Nama Produk\tHarga" << endl;

Node *current = head;

```

```

while (current != NULL)
{
    cout << current->nama << "\t\t" << current->harga
<< endl;
    current = current->next;
}
}

void updateDataAt(int index, string nama, int harga)
{
    if (index < 0 || index >= size)
    {
        cout << "Index out of bounds" << endl;
        return;
    }
    Node *current = head;
    for (int i = 0; i < index; i++)
    {
        current = current->next;
    }
    current->nama = nama;
    current->harga = harga;
}
};

int main()
{
    DoubleLinkedList dll;
    int choice;
    string nama;
    int harga;
    int index;
    do
    {
        cout << "Menu:" << endl;

```

```
cout << "1. Tambah Data" << endl;
    cout << "2. Hapus Data" << endl;
    cout << "3. Update Data" << endl;
    cout << "4. Tambah Data pada Urutan Tertentu" <<
endl;
    cout << "5. Hapus Data pada Urutan Tertentu" << endl;
    cout << "6. Hapus Semua Data" << endl;
    cout << "7. Tampilkan Data" << endl;
    cout << "8. Keluar" << endl;
    cout << "Pilih: ";
    cin >> choice;
    switch (choice)
    {
    case 1:
        cout << "Nama Produk: ";
        cin >> nama;
        cout << "Harga: ";
        cin >> harga;
        dll.addData(nama, harga);
        break;
    case 2:
        cout << "Index: ";
        cin >> index;
        dll.deleteDataAt(index);
        break;
    case 3:
        cout << "Index: ";
        cin >> index;
        cout << "Nama Produk: ";
        cin >> nama;
        cout << "Harga: ";
        cin >> harga;
        dll.updateDataAt(index, nama, harga);
```



```
break;

case 5:
    cout << "Index: ";
    cin >> index;
    dll.deleteDataAt(index);
    break;

case 6:
    dll.clearData();
    break;

case 7:
    dll.displayData();
    break;

case 8:
    break;

default:
    cout << "Pilihan tidak valid" << endl;
    break;

}

cout << endl;

//Output

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

} while (choice != 8);

return 0;

}
```

Output

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

```
Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 4
Index: 2
Nama Produk: Azarine
Harga: 65000

Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 7
Nama Produk      Harga
Originote        60000
Somethinc         150000
Azarine           65000
Skintific         100000
Wardah            50000
Hanasui           30000
```

- b. Hapus produk Wardah

```
Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 5
Index: 4

Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 7
Nama Produk      Harga
Originote        60000
Somethinc         150000
Azarine           65000
Skintific         100000
Hanasui           30000
```

c. Update produk Hanasui menjadi Cleora dengan harga 55000 dan tampilkan seluruh data

```
Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 3
Index: 4
Nama Produk: Cleora
Harga: 55000

Menu:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data pada Urutan Tertentu
5. Hapus Data pada Urutan Tertentu
6. Hapus Semua Data
7. Tampilkan Data
8. Keluar
Pilih: 7
Nama Produk      Harga
Originote        60000
Somethinc         150000
Azarine          65000
Skintific         100000
Cleora            55000
```

Deskripsi program

Program tersebut menyediakan operasi dasar untuk mengelola linked list berganda, seperti penambahan data, penghapusan data, pembaruan data, dan penampilan data. Setiap kali fungsi-fungsi ini dipanggil, program akan menampilkan menu dan menunggu masukan dari pengguna. Kemudian, program akan mengeksekusi tindakan yang dipilih berdasarkan masukan pengguna.

BAB V KESIMPULAN

Dibandingkan dengan single linked list yang hanya memiliki satu pointer untuk menghubungkan node secara berurutan, double linked list memiliki dua pointer yang menghubungkan node secara berurutan dan dua arah. Meskipun fitur tambahan ini memungkinkan lebih fleksibilitas saat menjalankan operasi seperti menambah atau menghapus node dengan mudah, ini juga memerlukan memori tambahan untuk menyimpan kedua pointer tersebut.

DAFTAR PUSTAKA

Menggunakan single dan double linked list pada c++

<https://ocw.upj.ac.id/files/Handout-TIF104-Modul-Praltikum-Struktur-Data.pdf>