

LAPORAN PRAKTIKUM

MODUL 3 SINGLE AND DOUBLE LINKED LIST



Disusun oleh:
Rakha Yudhistira
NIM: 2311102010

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

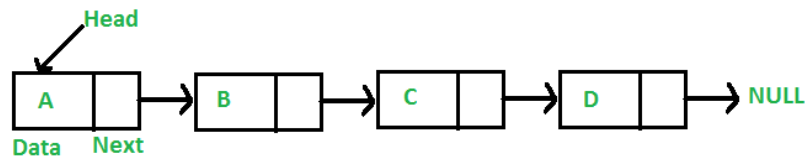
1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List.
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman.

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.



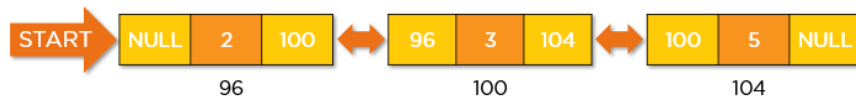
Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List umumnya 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.



c) Karakteristik Linked List

Sebuah linked list memiliki beberapa karakteristik sebagai berikut:

- Linked list menggunakan memori tambahan untuk menyimpan link (tautan)
- Untuk inisialisasi awal linked list, kita tidak perlu tahu ukuran dari elemen.
- Linked list umumnya dapat digunakan untuk mengimplementasikan struktur data lain seperti stack, queue, ataupun graf
- Simpul pertama dari linked list disebut sebagai Head.
- Pointer setelah simpul terakhir selalu bernilai NULL
- Dalam struktur data linked list, operasi penyisipan dan penghapusan dapat dilakukan dengan mudah
- Tiap-tiap simpul dari linked list berisi pointer atau tautan yang menjadi alamat dari simpul berikutnya
- Linked list bisa menyusut atau bertambah kapan saja dengan mudah.

Operasi-operasi pada Linked List

Ada beberapa operasi yang bisa kita lakukan pada struktur data linked list. Misalnya, operasi insertion yaitu tindakan menambahkan elemen baru ke linked list.

Berikut adalah daftar operasi dasar pada linked list:

- **Traversal** - mengakses setiap elemen dari linked list
- **Insertion** - menambahkan elemen baru ke linked list
- **Deletion** - menghapus elemen yang ada
- **Searching** - menemukan simpul pada linked list
- **Sorting** - mengurutkan simpul dari struktur linked list

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;
        }
    }
    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 << "\\t";
        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, "enam");
    tampil();
    ubahBelakang(8, "tujuh");
    tampil();
    ubahTengah(11, "delapan", 2);
    tampil();
    return 0;
}

```

Screenshoot program

```

3      satu
3      satu  5      dua
2      tiga  3      satu  5      dua
1      empat 2      tiga  3      satu  5      dua
2      tiga  3      satu
2      tiga  3      satu
2      tiga  7      lima  3      satu
2      tiga  3      satu
1      enam  3      satu
1      enam  8      tujuh
1      enam  11     tujuh

```

Deskripsi program

Program ini menggunakan pendekatan single linked list non-circular dengan hanya mengandalkan satu pointer ke node berikutnya, implementasi program ini memiliki dua pointer yang menunjuk ke node head dan tail dari linked list. Penggunaan Linked list dapat membuat program lebih kompleks dan efisien.

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 = 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->kata ==
oldKata)
            {
                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 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

```
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: 3
Enter kata to add: ppp
```

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

Deskripsi Program

Program ini mengimplementasikan struktur data Doubly Linked List dan menyediakan beberapa operasi seperti menambahkan data, menghapus data, mengubah data, menampilkan data dalam linked list, serta menghapus semua data di dalam linked list. Data yang diinputkan bertipe data int dan string untuk sebuah kata.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

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

class LinkedList
{
private:
    Node *head;

public:
    LinkedList()
    {
        head = nullptr;
    }

    void insertAwal(string nama, int usia)
    {
        Node *newNode = new Node;
        newNode->nama = nama;
        newNode->usia = usia;
        newNode->next = head;
        head = newNode;
    }
}
```

```

void insertAkhir(string nama, int usia)
{
    Node *newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;
    newNode->next = nullptr;

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

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

void insertSetelah(string nama, int usia, string
namaSebelum)
{
    Node *newNode = new Node;
    newNode->nama = nama;
    newNode->usia = usia;

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

```

```

        if (temp == nullptr)
        {
            cout << "Node dengan nama " << namaSebelum << " tidak
ditemukan." << endl;
            return;
        }

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

void hapus(string nama)
{
    if (head == nullptr)
    {
        cout << "Linked list kosong." << endl;
        return;
    }

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

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

```



```

    }

    if (temp == nullptr)
    {
        cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
        return;
    }

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

void ubah(string nama, string namaBaru, int usiaBaru)
{
    Node *temp = head;
    while (temp != nullptr && temp->nama != nama)
    {
        temp = temp->next;
    }

    if (temp == nullptr)
    {
        cout << "Node dengan nama " << nama << " tidak
ditemukan." << endl;
        return;
    }

    temp->nama = namaBaru;
    temp->usia = usiaBaru;
}

void tampilkan()
{

```

```

        Node *temp = head;
        while (temp != nullptr)
        {
            cout << temp->nama << " " << temp->usia << endl;
            temp = temp->next;
        }
    };

int main()
{
    LinkedList myList;
    myList.insertAwal("Rakha Yudhistira", 19);
    myList.insertAwal("John", 19);
    myList.insertAwal("Jane", 20);
    myList.insertAwal("Michael", 18);
    myList.insertAwal("Yusuke", 19);
    myList.insertAwal("Akechi", 20);
    myList.insertAwal("Hoshino", 18);
    myList.insertAwal("Karin", 18);

    cout << "Data setelah langkah (a):" << endl;
    myList.tampilkan();
    cout << endl;

    myList.hapus("Akechi");
    myList.insertSetelah("Futaba", 18, "John");
    myList.insertAwal("Igor", 20);
    myList.ubah("Michael", "Reyn", 18);
    cout << "Data setelah dilakukan semua operasi:" << endl;
    myList.tampilkan();

    return 0;
}

```

Screenshoot program

```
Data setelah langkah (a):  
Karin 18  
Hoshino 18  
Akechi 20  
Yusuke 19  
Michael 18  
Jane 20  
John 19  
Rakha Yudhistira 19  
  
Data setelah dilakukan semua operasi:  
Igor 20  
Karin 18  
Hoshino 18  
Yusuke 19  
Reyn 18  
Jane 20  
John 19  
Futaba 18  
Rakha Yudhistira 19
```

Deskripsi program

Program ini menggunakan struktur data linked list sederhana yang menyediakan berbagai opsi untuk pengguna memilih. Menu terdiri dari menambahkan data ke dalam linked list, memperbarui data yang sudah ada, menghapus data tertentu, membersihkan semua data yang disimpan dalam daftar dan menampilkan seluruh data yang ada di dalamnya. Data yang diinputkan berupa Nama dan juga umur dari nama orangnya.

2. Unguided 2

Modifikasi Guided Double Linked List dilakukan dengan penambahan operasi untuk menambah data, menghapus, dan update di tengah / di urutan tertentu yang diminta. Selain itu, buatlah agar tampilannya menampilkan Nama produk dan harga.

Nama Produk	Harga
Originote	60.000
Somethinc	150.000

Skintific	100.000
Wardah	50.000
Hanasui	30.000

Case:

1. Tambahkan produk Azarine dengan harga 65000 diantara Somethinc dan Skintific
2. Hapus produk wardah
3. Update produk Hanasui menjadi Cleora dengan harga 55.000
4. Tampilkan menu seperti dibawah ini

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***

Pada menu 7, tampilan akhirnya akan menjadi seperti dibawah ini :

Nama Produk	Harga
Originote	60.000
Somethinc	150.000
Azarine	65.000
Skintific	100.000
Cleora	55.000

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)
        {
            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 << "Toko Skincare Purwokerto" << 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 4:
        cout << "Index: ";
        cin >> index;
        cout << "Nama Produk: ";
        cin >> nama;
        cout << "Harga: ";
        cin >> harga;
        dll.addDataAt(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;
} while (choice != 8);
return 0;
}

```

Screenshoot program

Tampilan data awal

```
Toko Skincare Purwokerto
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
Originate         60000
Somethinc         150000
Skinific          100000
Wardah            50000
Hanasui           30000
```

Case 1:

```
Toko Skincare Purwokerto
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

Toko Skincare Purwokerto
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
Originate         60000
Somethinc         150000
Azarine           65000
Skinific          100000
Wardah            50000
Hanasui           30000
```

Case 2 :

```
Toko Skincare Purwokerto
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

Toko Skincare Purwokerto
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
Originate         60000
Somethinc         150000
Azarine           65000
Skinific          100000
Hanasui           30000
```

Case 3 :

```
Toko Skincare Purwokerto
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

Toko Skincare Purwokerto
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
Originate         60000
Somethinc         150000
Azarine           65000
Skinific          100000
Cleora            55000
```

Case 4 :

```
Toko Skincare Purwokerto
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
Originate         60000
Somethinc         150000
Azarine           65000
Skinific          100000
Cleora            55000
```

Deskripsi program

Program ini dibuat untuk sebuah toko skincare dengan menggunakan operasi dasar untuk mengelola Doubly Linked List, 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 hal ini karena terdapat loop. Kemudian, program akan mengeksekusi tindakan yang dipilih berdasarkan masukan pengguna.

BAB IV

KESIMPULAN

Linked list adalah struktur data linier berbentuk rantai simpul di mana setiap simpul menyimpan 2 item, yaitu nilai data dan pointer ke simpul elemen berikutnya. Berbeda dengan array, elemen linked list tidak ditempatkan dalam alamat memori yang berdekatan melainkan elemen ditautkan menggunakan pointer.

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

Asisten Praktikum. (2024). MODUL 3 SINGLE AND DOUBLE LINKED LIST, Learning Management System

Trivusi. (16 September 2022). Struktur Data Linked List: Pengertian, Karakteristik, dan Jenis-jenisnya, Diakses pada 01 April 2024 dari <https://www.trivusi.web.id/2022/07/struktur-data-linked-list.html>