LAPORAN PRAKTIKUM STRUKTUR DATA DAN PEMROGRAMAN

MODUL 3

“SINGLE DAN DOUBLE LINKED LIST”

Disusun oleh :

Baharuddin Barkah Pratama

2311102321

IF-11-A

Dosen Pengampu :

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

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

A. Tujuan

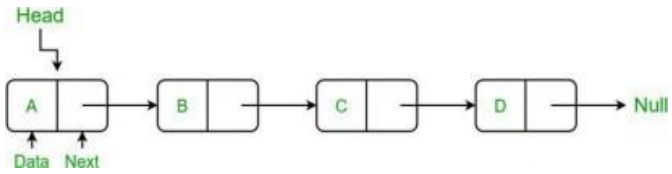
1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman

BAB II
DASAR TEORI

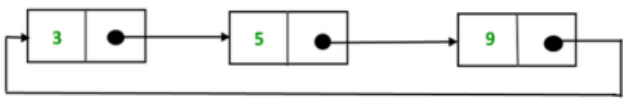
B. Dasar Teori

a) Single Linked List

Linked List adalah struktur data yang berisi kumpulan data yang disebut node, yang disusun secara berurutan dan saling terhubung satu sama lain melalui pointer. Struktur ini bersifat dinamis dan tidak terbatas. Setiap elemen dalam Linked List terhubung dengan elemen lain melalui pointer, di mana setiap elemen memiliki dua bagian utama. Bagian pertama adalah bagian informasi atau data yang menyimpan nilai dari elemen tersebut, sedangkan bagian kedua adalah bagian pointer yang menunjukkan alamat dari node berikutnya atau sebelumnya. Linked List terbentuk dengan cara menghubungkan pointer next suatu elemen ke elemen berikutnya. Elemen terakhir dari Linked List memiliki pointer next yang menunjukkan NULL, menandakan akhir dari list tersebut. Elemen pertama dari Linked List disebut head, sedangkan elemen terakhir disebut tail.



Dalam operasi Single Linked List, biasanya melibatkan penambahan dan penghapusan simpul di awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya menggunakan 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. Circular Linked List adalah jenis kedua dari Single Linked List. Perbedaan antara circular linked list dan non-circular linked list adalah bahwa penunjuk next pada node terakhir pada circular linked list akan selalu merujuk kembalike node pertama.

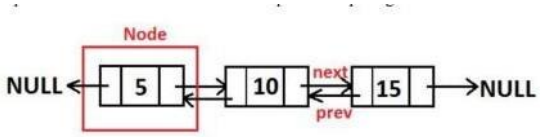


b) Double Linked List

Double Linked List adalah struktur data Linked List yang serupa dengan Single Linked List, tetapi dengan tambahan satu pointer tambahan pada setiap simpul, yaitu pointer prev yang menunjuk ke simpul sebelumnya. Dengan adanya pointer prev, Double Linked List memungkinkan operasi penghapusan dan penambahan pada simpul mana pun dengan efisien. Setiap simpul dalam Double Linked List memiliki tiga elemen kunci, yaitu data (umumnya berupa nilai), pointer next yang menunjuk ke simpul berikutnya, dan pointer prev yang menunjuk ke simpul sebelumnya.

Double Linked List memiliki keunggulan dalam melakukan operasi penghapusan dan penambahan pada simpul di mana pun dengan efisien, yang berguna dalam berbagai implementasi algoritma yang memerlukan operasi tersebut. Selain itu, Double Linked List memungkinkan traversal dari depan (head) maupun dari belakang (tail) dengan mudah. Namun, kelemahan dari Double Linked List adalah penggunaan memori yang lebih besar daripada Single Linked List karena setiap simpul membutuhkan satu pointer tambahan. Selain itu, waktu eksekusi dalam operasi penambahan dan penghapusan pada Double Linked List cenderung lebih lama dibandingkan dengan Single Linked List.

Representasi sebuah double linked list dapat dilihat pada gambar berikut ini:



Dalam sebuah linked list, terdapat dua pointer utama, yaitu pointer HEAD yang mengarah ke node pertama di dalam linked list dan pointer TAIL yang menunjuk ke node terakhir di dalam linked list tersebut. Sebuah linked list dianggap kosong jika pointer head-nya adalah NULL. Selain itu, nilai pointer prev dari HEAD selalu NULL karena merupakan data pertama. Demikian pula, pointer next dari TAIL selalu bernilai NULL sebagai penanda data terakhir.

BAB III
GUIDED

C. GUIDED

GUIDED 1

a. Latihan Single Linked List

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->kata = kata;
        baru->data = nilai;
        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, int posisi, string kata)
    {
        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)
    {
        if (isEmpty() == false)
        {
            head->data = data;
        }
        else
        {
            cout << "List masih kosong!" << endl;
        }
    }
    // Ubah Tengah
    void ubahTengah(int data, 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;
            }
        }
        else
        {
            cout << "List masih kosong!" << endl;
```

```
        }
    }
    // Ubah Belakang
    void ubahBelakang(int data)
    {
        if (isEmpty() == false)
        {
            tail->data = data;
        }
        else
        {
            cout << "List masih kosong!" << endl;
        }
    }
    // Hapus List
    void clearList()
    {
        Node *bantu, *hapus;
        bantu = head;
        while (bantu != NULL)
        {
            hapus = bantu;
            bantu = bantu->next;
            delete hapus;
        }
        head = tail = NULL;
        cout << "List berhasil terhapus!" << endl;
    }
    // Tampilkan List
    void tampil()
    {
        Node *bantu;
        bantu = head;
        if (isEmpty() == false)
        {
            while (bantu != NULL)
            {
                cout << bantu->data << "\t";
                cout << bantu->kata;
                bantu = bantu->next;
            }
            cout << endl;
        }
        else
        {
            cout << "List masih kosong!" << endl;
        }
    }
    int main()
    {
        init();
        insertDepan(3, "satu");
        tampil();
        insertBelakang(5 , "tiga");
        tampil();
        insertDepan(2 , "lima");
        tampil();
        insertDepan(1, "dua");
        tampil();
        hapusDepan();
        tampil();
```

```
            hapusBelakang();
            tampil();
            insertTengah(7, 2 , "tujuh");
            tampil();
            hapusTengah(2);
            tampil();
            ubahDepan(1);
            tampil();
            ubahBelakang(8);
            tampil();
            ubahTengah(11, 2);
            tampil();
            return 0;
}
```

SCREENSHOOT PROGRAM

DESKRIPSI PROGRAM

Program ini memiliki fungsi-fungsi untuk melakukan berbagai operasi pada linked list seperti menambahkan node di depan atau belakang, menghapus node di depan atau belakang, menambah atau menghapus node pada posisi tertentu, serta mengubah nilai dari node di depan, belakang, atau pada posisi tertentu. Selain itu, terdapat juga fungsi untuk menghitung jumlah node dalam linked list dan menampilkan isi linked list. Program ini menggunakan struct Node yang memiliki tiga komponen yaitu integer, string, dan pointer next untuk menunjuk ke node selanjutnya. Di dalam fungsi main, ada contoh penggunaan dari setiap fungsi yang telah didefinisikan sebelumnya, mulai dari penambahan, penghapusan, pengubahan nilai, dan penampilan isi linked list. Program ini memberikan fleksibilitas dalam manipulasi data dalam linked list dengan berbagai operasi yang disediakan.

GUIDED 2

b. Latihan Double Linked List

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 :

DoublyLinkedList adalah kelas yang mengimplementasikan linked list. Kelas ini memiliki dua pointer yaitu head yang menunjuk ke node pertama dan tail yang menunjuk ke node terakhir. Metode push() menambahkan node baru di depan linked list dengan mengatur pointer prev dan next sesuai penambahan. Metode pop() berfungi untuk menghapus node dari depan linked list. Metode update() untuk mengganti nilai data dan kata pada node dengan nilai baru, jika data lama ditemukan. Metode deleteAll() untuk menghapus semua node dari linked list. Metode display() akan menampilkan isi linked list. Fungsi main() memiliki loop untuk memilih operasi

berupa menambah, menghapus, mengubah, membersihkan, atau menampilkan isi linked list. Pengguna memilih opsi dan program menjalankan operasi sesuai pilihan hingga pengguna memilih untuk keluar.

BAB IV

UNGUIDED

UNGUIDED 1

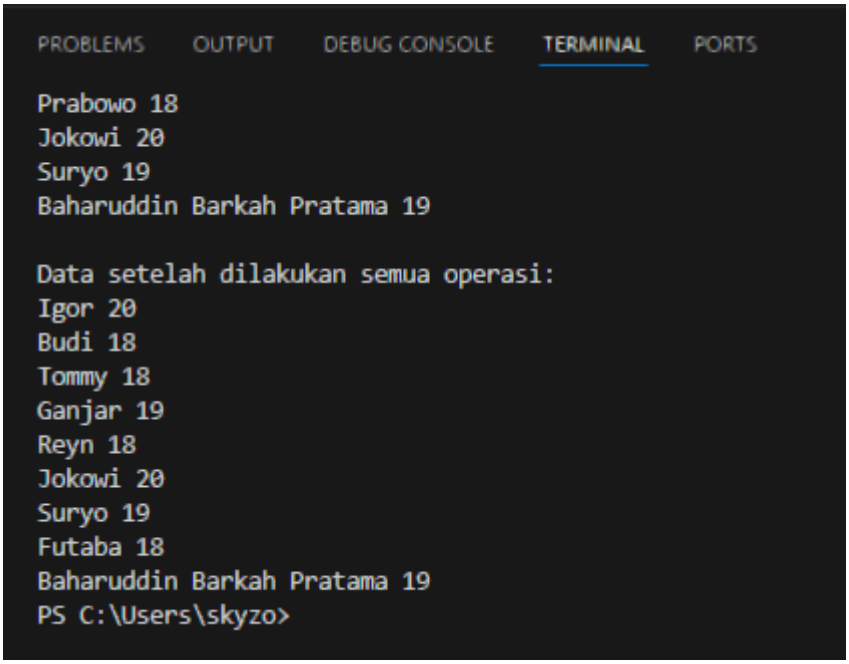
SOURCE CODE

```
#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("Baharuddin Barkah Pratama", 19 );
    myList.insertAwal("Suryo", 19);
    myList.insertAwal("Jokowi", 20);
    myList.insertAwal("Prabowo", 18);
    myList.insertAwal("Ganjar", 19);
    myList.insertAwal("Naruto", 20);
    myList.insertAwal("Tommy", 18);
    myList.insertAwal("Budi", 18);

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

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

    return 0;
}
```

SCREENSHOOT PROGRAM



Pertama, data dimasukkan ke dalam linked list dengan menggunakan “operasi insertAwal” untuk memasukkan data ke awal list. Kemudian, data "Naruto" dihapus dari list dengan menggunakan operasi “ hapus’ . Selanjutnya, data "Futaba" dimasukkan setelah data "John" menggunakan operasi “insertSetelah”. Data "Igor" dimasukkan ke awal list dengan menggunakan operasi “insertAwal” . Akhirnya, data "Michael" diubah menjadi "Reyn" dengan

usia 18 menggunakan operasi “ubah”. Setelah semua operasi dilakukan, isi linked list ditampilkan menggunakan operasi “tampilkan”. Output program menampilkan data yang dimasukkan dan diubah sesuai dengan langkah-langkah tersebut.

UNGUIDE 2

SOURCE KODE

```
#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 tidak di temukan" << 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 << "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 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

1.Tambahkan produk SisterLemon dengan harga 5000 diantara MamaLemon dan BabyLemon

```
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: SisterLemon
Harga: 5000

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
PapaLemon               6500
MamaLemon               4000
SisterLemon             5000
BabyLemon               4000
```

2.Hapus Produk BabyLemon

```
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: 2
Index: 3

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
PapaLemon               6500
MamaLemon               4000
SisterLemon             5000
```

3.Update produk PapaLemon menjadi Lifebuoy dengan harga 3.500, tampilkan seperti nomer 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: 3
Index: 0
Nama Produk: Lifebuoy
Harga: 3500

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
Lifebuoy                3500
MamaLemon               4000
SisterLemon             5000
```

DESKRIPSI PROGRAM

Pengguna akan diberi opsi melalui menu untuk memilih tindakan. Setelah memilih, mereka diminta untuk memasukkan informasi yang dibutuhkan. Misalnya, untuk menambah data, mereka diminta nama produk dan harganya. Setelah selesai, pengguna dapat keluar dari program. Output akan menampilkan data sesuai tindakan pengguna, seperti menambah, menghapus, memperbarui, atau menghapus semua data. Jika ada kesalahan input, pesan kesalahan akan ditampilkan. Program berjalan dalam loop hingga pengguna keluar.

KESIMPULAN

Dari semua kode yang telah disediakan, dapat disimpulkan bahwa kedua jenis linked list, yaitu single linked list dan double linked list, digunakan untuk menyimpan dan mengelola data dalam urutan terhubung. Single linked list memiliki setiap node yang terhubung hanya ke node berikutnya, sementara double linked list memiliki setiap node yang terhubung ke node sebelumnya dan berikutnya.

DAFTAR PUSTAKA

https://www.softwaretestinghelp.com/doubly-linked-list-2/