LAPORAN PRAKTIKUM

MODUL III

SINGLE DAN DOUBLE LINKED LIST

Disusun Oleh:

ALDI RANSI

2211102328

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

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

BAB I

TUJUAN PRAKTIKUM

- Mampu memahami perbedaan konsep Single dan Double Linked List..
- Mampu mampu menerapkan Single dan Double Linked List ke dalam pemrograman

BAB II

DASAR TEORI

1. Linked List

   Linked list banyak digunakan sebagai struktur data dasar dalam ilmu komputer. Ide dasarnya berkisar pada pengelompokan atau vertex yang disebut simpul atau node. Simpul-simpul ini memiliki dua komponen utama: data sebenarnya dan 'pointer' yang menghubungkannya dengan elemen atau node berikutnya dalam rangkaian matematisnya. Head mewakili langkah pertama sedangkan Tail menandakan langkah terakhir.

2. Single Linked List

   Single linked list adalah struktur yang terdiri dari simpul-simpul yang dikelompokkan secara berurutan. Setiap simpul berisi dua elemen penting, yaitu data yang disimpan serta indikasi simpul mana yang mengikutinya. Pada akhirnya, fitur ini membantu pengguna menambah atau menghapus informasi yang diperlukan tanpa mengganggu integritas keseluruhan dari dataset mereka.

3. Double Linked List

   Struktur data double linked list memiliki ciri-ciri terdiri dari simpul-simpul yang terhubung secara berurutan dan dua arah. Setiap simpul dalam double linked list memiliki dua pointer, yaitu pointer ke simpul sebelumnya dan pointer ke simpul berikutnya dalam urutan. Dengan fitur ini, pengguna dapat melakukan operasi penambahan atau penghapusan simpul dengan lebih mudah dan fleksibel dibandingkan dengan single linked list. Namun, struktur data ini memerlukan lebih banyak penggunaan memori karena menyimpan kedua pointer tersebut.

BAB III

GUIDED

1. Guided 1

Source code

```
#include <iostream>
using namespace std;
/// PROGRAM SINGLE LINKED LIST NON-CIRCULAR
// Deklarasi Struct Node

struct Node
{
    int data;
    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)
{
    // Buat Node baru
    Node *baru = new Node;
    baru->data = nilai;
    baru->next = NULL;

    if (isEmpty() == true)
    {
        head = tail = baru;
        tail->next = NULL;
    }
```

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

// Tambah Belakang
void insertBelakang(int nilai)
{
    // Buat Node baru
    Node *baru = new Node;
    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)
{
    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;

        // 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 << ends;
            bantu = bantu->next;
        }
        cout << endl;
    }
    else
    {
        cout << "List masih kosong!" << endl;
    }
}
```

```
int main()
{
    init();
    insertDepan(3);

    tampil();
    insertBelakang(5);
    tampil();
    insertDepan(2);
    tampil();
    insertDepan(1);
    tampil();
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();
    insertTengah(7, 2);
    tampil();
    hapusTengah(2);
    tampil();
    ubahDepan(1);
    tampil();
    ubahBelakang(8);
    tampil();
    ubahTengah(11, 2);
    tampil();

    return 0;
}
```

Output

```
PS C:\Users\ACER>  & 'c:\Users\ACER\.vscode\extensio
ns\ms-vscode.cpptools-1.14.5-win32-x64\debugAdapters
\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MI
Engine-In-gnhshg4a.r1h' '--stdout=Microsoft-MIEngine
-Out-gcrunjpi.kyv' '--stderr=Microsoft-MIEngine-Erro
r-234shyzo.y2c' '--pid=Microsoft-MIEngine-Pid-uoup51
z4.vqr' '--dbgExe=C:\Not System\College\MinGW\bin\gd
b.exe' '--interpreter=mi'
3
35
235
1235
235
23
273
23
13
18
111
PS C:\Users\ACER>
```

Deskripsi program

Pada Program tersebut menggunakan pendekatan single linked list non-circular dengan hanya mengandalkan satu pointer ke node berikutnya dan tidak memiliki loop, implementasi program ini memiliki dua pointer yang menunjukke node head dan tail dari linked list.

2. Guided 2

Source code

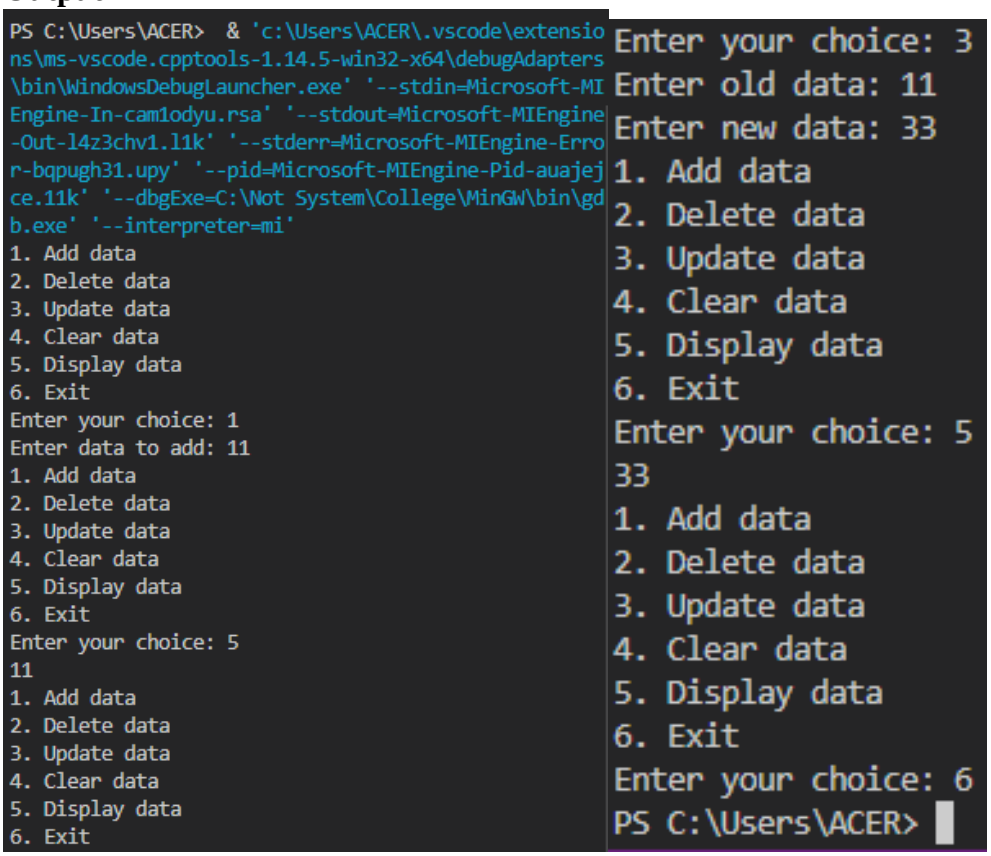
```
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node *prev;
    Node *next;
};
class DoublyLinkedList
{
public:
    Node *head;
    Node *tail;
    DoublyLinkedList()
    {
        head = nullptr;
```

```
        tail = nullptr;
    }
    void push(int data)
    {
        Node *newNode = new Node;
        newNode->data = data;
        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)
    {
        Node *current = head;

        while (current != nullptr)
        {
            if (current->data == oldData)
            {
                current->data = newData;
                return true;
            }
            current = current->next;
        }
        return false;
    }
    void deleteAll()
```

```
    {
        Node *current = head;
        while (current != nullptr)
        {
            Node *temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }
    void display()
    {
        Node *current = head;
        while (current != nullptr)
        {
            cout << current->data << " ";
            current = 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;
            cout << "Enter data to add: ";
            cin >> data;
            list.push(data);
            break;
        }
        case 2:
        {
            list.pop();
            break;
        }
```

```
        case 3:
        {
            int oldData, newData;
            cout << "Enter old data: ";
            cin >> oldData;
            cout << "Enter new data: ";
            cin >> newData;
            bool updated = list.update(oldData, newData);
            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



Deskripsi program

Program tersebut 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.

UNGUIDED

1. **Unguided 1**
   Source code

```
#include <iostream>
// Muhammad Rayhan Pratama/2211102179/IF-10-D
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;
```

```
            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. Masukkan data sesuai urutan

```
MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Rayhan, Age: 18
Name: Budi, Age: 19
Name: Carol, Age: 20
Name: Ann, Age: 18
Name: Yusuke, Age: 19
Name: Akechi, Age: 20
Name: Hoshino, Age: 18
Name: Karin, Age: 18
```

b. Hapus data Akechi

```
MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 3
Enter name to delete: Akechi

MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Rayhan, Age: 18
Name: Budi, Age: 19
Name: Carol, Age: 20
Name: Ann, Age: 18
Name: Yusuke, Age: 19
Name: Hoshino, Age: 18
Name: Karin, Age: 18
```

c. Tambahkan data berikut diantara Carol dan Ann

```
MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Rayhan, Age: 18
Name: Budi, Age: 19
Name: Carol, Age: 20
Name: Futaba, Age: 18
Name: Ann, Age: 18
Name: Yusuke, Age: 19
Name: Hoshino, Age: 18
Name: Karin, Age: 18
```

d. Tambahkan data berikut di awal

```
MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 1
Enter name: Igor
Enter age: 20

MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Igor, Age: 20
Name: Rayhan, Age: 18
Name: Budi, Age: 19
Name: Carol, Age: 20
Name: Futaba, Age: 18
Name: Ann, Age: 18
Name: Yusuke, Age: 19
Name: Hoshino, Age: 18
Name: Karin, Age: 18
```

e. Ubah data Carol dan tampilkan seluruh data

```
MENU
1. Add data
2. Update data
3. Delete data
4. Clear data
5. Display data
6. Exit
Enter your choice: 5
Name: Igor, Age: 20
Name: Rayhan, Age: 18
Name: Budi, Age: 19
Name: Reyn, Age: 18
Name: Futaba, Age: 18
Name: Ann, Age: 18
Name: Yusuke, Age: 19
Name: Hoshino, Age: 18
Name: Karin, Age: 18
```

Deskripsi program

Program tersebut terdapat Sebuah 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.

2. Unguided 2

Source code

```
#include <iostream>
// Muhammad Rayhan Pratama/2211102179/IF-10-D
#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 << "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;
}
```

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 maka program akan menampilkan menu dan menunggu masukan dari pengguna. Kemudian, program akan mengeksekusi tindakan yang dipilih berdasarkan masukan pengguna.

BAB IV
KESIMPULAN

Dari hasil praktikuk yang telah di lakukan dapat di ambil kesimpulan bahwa Dibandingkan dengan single linked list yang hanya memiliki satu pointer saja 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 tetapi hal ini juga memerlukan memoritambahan untuk menyimpan kedua pointer tersebut.