

LAPORAN PRAKTIKUM

MODUL III SINGLE & DOUBLE LINKED LIST



**Disusun oleh:
Aryo Tegar Sukarno
NIM: 2311102018**

**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

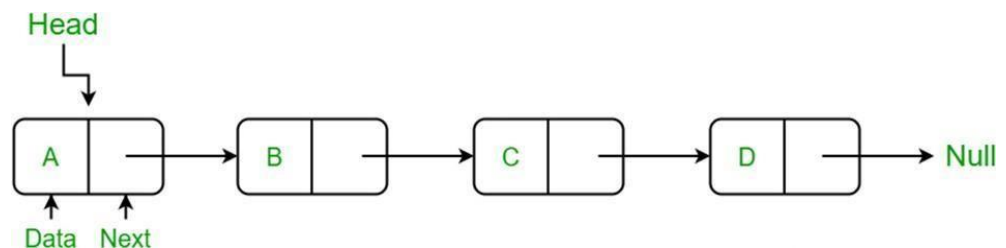
1. Mahasiswa memahami perbedaan konsep Single dan Double Linked List
2. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemrograman.

BAB II

DASAR TEORI

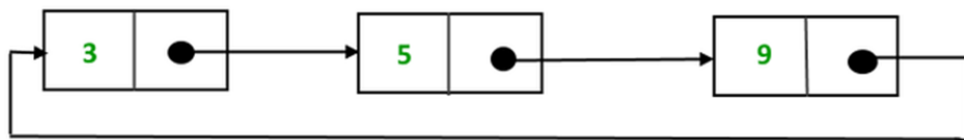
a. Single Linked List

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.

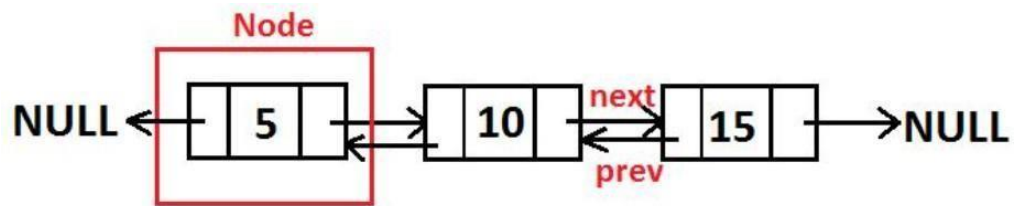


b. Double Linked List

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

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



Di dalam sebuah linked list, ada 2 pointer yang menjadi penunjuk utama, yakni pointer HEAD yang menunjuk pada node pertama di dalam linked list itu sendiri dan pointer TAIL yang menunjuk pada node paling akhir di dalam linked list. Sebuah linked list dikatakan kosong apabila isi pointer head adalah NULL. Selain itu, nilai pointer prev dari HEAD selalu NULL, karena merupakan data pertama. Begitu pula dengan pointer next dari TAIL yang selalu bernilai NULL sebagai penanda data terakhir.

BAB III

GUIDED

1. Guided 1

Source code

```
#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

```
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 3\MODUL3> cd 'c:\Users\
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 3\MODUL3\output> & .\
3      satu
3      satu 5      dua
2      tiga 3      satu 5      dua
1      empat 2      tiga 3      satu 5      dua
2      tiga 3      satu 5      dua
2      tiga 3      satu
2      tiga 7      lima 3      satu
2      tiga 3      satu
1      enam 3      satu
1      enam 8      tujuh
1      enam 11     tujuh
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 3\MODUL3\output>
```

Deskripsi program

Program di atas merupakan implementasi dari sebuah single linked list non-circular dalam bahasa pemrograman C++. Dalam program tersebut, dilakukan pendefinisian sebuah struktur data Node yang memiliki komponen integer (data), string (kata), dan pointer ke Node berikutnya. Di dalam fungsi `main()`, dilakukan pengujian terhadap fungsi-fungsi tersebut dengan menambah, menghapus, mengubah, dan menampilkan elemen-elemen dalam list.

2. Guided 2

Source code

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

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

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

```

Screenshot program


```
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 3\MODUL3> cd 'c:\Users\aryos\Down
PS C:\Users\aryos\Downloads\PROKTIKUM SEMESTER 2\MODUL 3\MODUL3\output> & .\'guided2.ex
1. Add data
2. Delete data
3. Update data
4. Clear data
5. Display data
6. Exit
Enter your choice: █
```

Deskripsi program

Program di atas adalah implementasi dari doubly linked list menggunakan bahasa pemrograman C++. Doubly linked list adalah struktur data linear di mana setiap elemen terdiri dari sebuah node yang menyimpan data, dua pointer yang menunjuk ke node sebelumnya (prev) dan node selanjutnya (next). Dengan ini, pengguna dapat berinteraksi dengan doubly linked list, menambahkan, menghapus, mengubah data, membersihkan isi list, dan menampilkan isi list sesuai kebutuhan.

LATIHAN KELAS - UNGUIDED

1. Unguided 1

Source code

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

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

    Node(string _name, int _age) {
        name = _name;
        age = _age;
        next = NULL;
        prev = NULL;
    }
}
```

```

};

class DoublyLinkedList {
public:
    Node* head;

    DoublyLinkedList() {
        head = NULL;
    }

    // Fungsi untuk menambahkan node di depan
    void insertAtFront(string name, int age) {
        Node* newNode = new Node(name, age);
        if (head != NULL) {
            newNode->next = head;
            head->prev = newNode;
        }
        head = newNode;
    }

    // Fungsi untuk menambahkan node di belakang
    void insertAtEnd(string name, int age) {
        Node* newNode = new Node(name, age);
        if (head == NULL) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != NULL) {
                temp = temp->next;
            }
            temp->next = newNode;
            newNode->prev = temp;
        }
    }

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

```

```

void deleteNode(string name) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->name == name) {
            if (temp == head) {
                head = temp->next;
                if (head != NULL) {
                    head->prev = NULL;
                }
            } else {
                temp->prev->next = temp->next;
                if (temp->next != NULL) {
                    temp->next->prev = temp->prev;
                }
            }
            delete temp;
            return;
        }
        temp = temp->next;
    }
    cout << name << " 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 clearAll() {
    Node* temp = head;
    while (temp != NULL) {
        Node* next = temp->next;
        delete temp;
    }
}

```

```

        temp = next;
    }
    head = NULL;
}

};

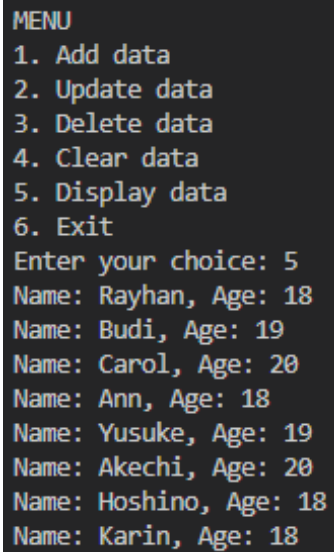
int main() {
    DoublyLinkedList list;
    int choice;
    string name;
    int age;
    do {
        cout << endl;
        cout << "MENU" << endl;
        cout << "1. Add data" << endl;
        cout << "2. Update data" << endl;
        cout << "3. Delete data" << endl;
        cout << "4. Clear data" << 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;
}
```

Screenshoot program

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

3. Source code

```
4. #include <iostream>
5. using namespace std;
6.
7.
8. class Node {
9. public:
```



```
10.         string productName;
11.         int price;
12.         Node* next;
13.         Node* prev;
14.
15.         Node(string _productName, int _price) {
16.             productName = _productName;
17.             price = _price;
18.             next = NULL;
19.             prev = NULL;
20.         }
21.     };
22.
23.
24.     class DoubleLinkedList {
25.     public:
26.         Node* head;
27.
28.         DoubleLinkedList() {
29.             head = NULL;
30.         }
31.
32.
33.         void insertFront(string productName, int price) {
34.             Node* newNode = new Node(productName, price);
35.             if (head != NULL) {
36.                 newNode->next = head;
37.                 head->prev = newNode;
38.             }
39.             head = newNode;
40.         }
41.
42.
43.         void insertEnd(string productName, int price) {
44.             Node* newNode = new Node(productName, price);
45.             if (head == NULL) {
46.                 head = newNode;
47.             } else {
48.                 Node* temp = head;
49.                 while (temp->next != NULL) {
50.                     temp = temp->next;
51.                 }
52.                 temp->next = newNode;
53.                 newNode->prev = temp;
```

```

54.         }
55.     }
56.
57.
58.     void insertAfter(string productName, int price,
string keyProductName) {
59.         Node* temp = head;
60.         while (temp != NULL) {
61.             if (temp->productName == keyProductName) {
62.                 Node* newNode = new Node(productName,
price);
63.                 newNode->next = temp->next;
64.                 if (temp->next != NULL) {
65.                     temp->next->prev = newNode;
66.                 }
67.                 temp->next = newNode;
68.                 newNode->prev = temp;
69.                 return;
70.             }
71.             temp = temp->next;
72.         }
73.         cout << keyProductName << " not found in the
list." << endl;
74.     }
75.
76.
77.     void deleteNode(string productName) {
78.         Node* temp = head;
79.         while (temp != NULL) {
80.             if (temp->productName == productName) {
81.                 if (temp == head) {
82.                     head = temp->next;
83.                     if (head != NULL) {
84.                         head->prev = NULL;
85.                     }
86.                 } else {
87.                     temp->prev->next = temp->next;
88.                     if (temp->next != NULL) {
89.                         temp->next->prev = temp->prev;
90.                     }
91.                 }
92.                 delete temp;
93.                 return;
94.             }

```

```

95.             temp = temp->next;
96.         }
97.         cout << productName << " not found in the list."
    << endl;
98.     }
99.
100.
101.     void updateNode(string productName, int price) {
102.         Node* temp = head;
103.         while (temp != NULL) {
104.             if (temp->productName == productName) {
105.                 temp->price = price;
106.                 return;
107.             }
108.             temp = temp->next;
109.         }
110.         cout << productName << " not found in the list."
    << endl;
111.     }
112.
113.
114.     void display() {
115.         Node* temp = head;
116.         cout << "Nama Produk\tHarga" << endl;
117.         while (temp != NULL) {
118.             cout << temp->productName << "\t\t" << temp-
    >price << endl;
119.             temp = temp->next;
120.         }
121.     }
122. };
123.
124. int main() {
125.     DoubleLinkedList productList;
126.
127.     // Menambahkan data awal
128.     productList.insertEnd("Originote", 60000);
129.     productList.insertEnd("Somethinc", 150000);
130.     productList.insertEnd("Skintific", 100000);
131.     productList.insertEnd("Wardah", 50000);
132.     productList.insertEnd("Hanasui", 30000);
133.
134.     int choice;
135.     string productName, keyProductName;

```



```

174.                cin >> price;
175.                cout << "Masukkan nama produk
    setelahnya: ";
176.                cin >> keyProductName;
177.                productList.insertAfter(productName,
    price, keyProductName);
178.                break;
179.            case 5:
180.                // Tidak diimplementasikan sesuai
    permintaan
181.                cout << "Fitur belum
    diimplementasikan." << endl;
182.                break;
183.            case 6:
184.                productList.display();
185.                break;
186.            case 7:
187.                cout << "Exiting program..." << endl;
188.                break;
189.            default:
190.                cout << "Pilihan tidak valid." << endl;
191.        }
192.    } while (choice != 8);
193.
194.    return 0;
195.    }

```

Screenshoot program

- 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 IV

KESIMPULAN

Single dan double linked list adalah dua struktur data penting dalam pemrograman, yang digunakan untuk menyimpan dan mengorganisir kumpulan data secara terurut. Kesimpulannya, pemilihan antara single dan double linked list tergantung pada kebutuhan dan kinerja aplikasi yang diinginkan. Single linked list cenderung lebih hemat ruang memori dan cocok untuk aplikasi yang hanya memerlukan penambahan dan penghapusan di bagian depan atau belakang list. Sementara itu, double linked list lebih fleksibel dalam operasi penambahan, penghapusan, dan traversal maju-mundur, namun membutuhkan lebih banyak ruang memori.