LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL III
SINGLE AND DOUBLE LINKED LIST

Disusun Oleh :

NAMA : M.Fathiakmal.L.A

NIM :18102096

Dosen

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

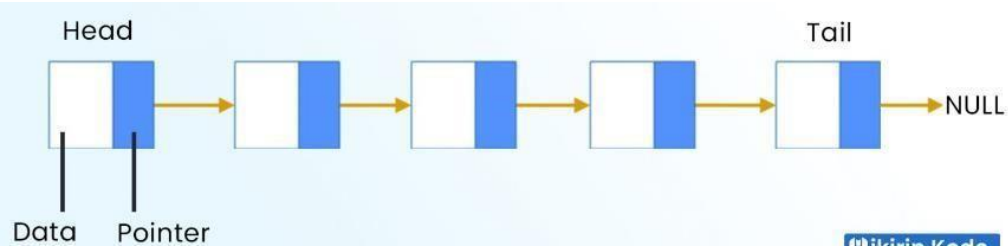
PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFROMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024

A. Dasar Teori

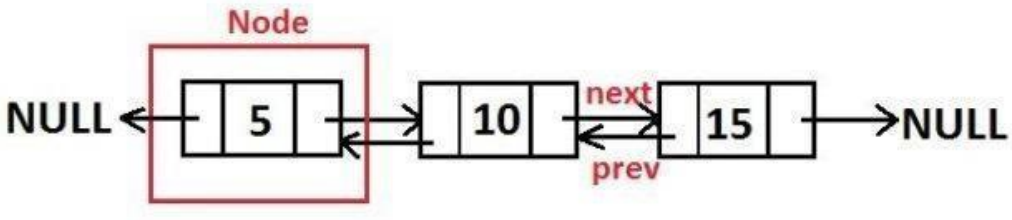
a. Single Linked List

Single Linked List atau Singly Linked List merupakan linked list yang hanya memiliki satu variabel pointer untuk menunjuk ke node lainnya, variabel pointer ini biasanya dinamakan next. Pada dasarnya Single Linked List ini adalah linked list yang berbentuk umum seperti yang dijelaskan sebelumnya. Seperti yang dapat dilihat dari ilustrasi di atas, terdapat 5 node yang terhubung menjadi suatu single linked list. Node pertama biasa disebut head, pointer pada node ini menunjuk ke node selanjutnya dan begitu seterusnya hingga node terakhir yang pointernya menujuk ke NULL. Hal itu menandakan bahwa node itu adalah node terakhir atau biasa disebut dengan node tail.



b. Double Linked List

Dalam pembahasan artikel sebelumnya telah diperkenalkan Single Linked List, yakni linked list dengan sebuah pointer penghubung. Dalam artikel ini, dibahas pula varian linked list dengan 2 pointer penunjuk, yakni Doubly linked list yang memilki pointer penunjuk 2 arah, yakni ke arah node sebelum (previos/prev) dan node sesudah (next). Representasi sebuah doubly 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.

B. Guided

Guided 1

```
#include <iostream> using
namespace std;
struct
Node
{ int data;
    Node *next;
};

Node *head;
Node *tail;
void
init()
{  head  =  NULL;
    tail = NULL;
}
bool
isEmpty()
{ return head == NULL;
}
void insertDepan(int
nilai)
{
    Node *baru = new Node;
    baru->data   =   nilai;
    baru->next  =  NULL;  if
    (isEmpty())
    { head = tail = baru;
    }
    else
    {  baru->next  =  head;
        head = baru;
    }
}
void insertBelakang(int
nilai)
```

```
{
    Node *baru = new Node;
    baru->data   =  nilai;
    baru->next = NULL; if
    (isEmpty())
    { head = tail = baru;
    }
    else
    {  tail->next  =  baru;
        tail = baru;
    }
}
 int
hitungList()
{
    Node  *hitung  =  head;
    int jumlah = 0; while
    (hitung != NULL)
    { jumlah++; hitung =
        hitung->next;
    } return
    jumlah;
}
 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  =  new  Node();
        baru->data  =  data;  Node
        *bantu = head; int nomor =
        1; while (nomor < posisi -
        1)
```

```
        { bantu = bantu->next;
            nomor++;
        }  baru->next  =  bantu-
        >next;   bantu->next   =
        baru;
    }
}
void
hapusDepan()
{ if (!isEmpty())
    {
        Node *hapus = head; if
        (head->next != NULL)
        {  head  =  head->next;
            delete hapus;
        }
        else
        { head = tail = NULL;
            delete hapus;
        }
    }
    else
    { cout << "List kosong!" << endl;
    }
}
void
hapusBelakang()
{ if (!isEmpty())
    { if (head != tail)
        {
            Node *hapus = tail; Node
            *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;
    }
}
void hapusTengah(int
posisi)
{ if (posisi < 1 || posisi > hitungList())
    { cout << "Posisi diluar jangkauan" << endl;
    } else if (posisi ==
    1)
    { cout << "Posisi bukan posisi tengah" << endl;
    }
    else
    {
        Node *hapus; Node
        *bantu = head;
        for (int nomor = 1; nomor < posisi - 1; nomor++)
        { bantu = bantu->next;
        }
        hapus    =    bantu->next;
        bantu->next = hapus->next;
        delete hapus;
    }
}
void ubahDepan(int
data)
{ if (!isEmpty())
    { head->data = data;
    }
```

```
    else
    { cout << "List masih kosong!" << endl;
    }
}
void ubahTengah(int data, int
posisi)
{ if (!isEmpty())
    { if (posisi < 1 || posisi > hitungList())
        { cout << "Posisi di luar jangkauan" << endl;
        } else if (posisi ==
        1)
        { cout << "Posisi bukan posisi tengah" << endl;
        }
        else
        {
            Node *bantu = head; for (int nomor = 1;
            nomor < posisi; nomor++)
            { bantu = bantu->next;
            } bantu->data =
            data;
        }
    }
    else
    { cout << "List masih kosong!" << endl;
    }
}
void ubahBelakang(int
data)
{ if (!isEmpty())
    { tail->data = data;
    }
    else
    { cout << "List masih kosong!" << endl;
    }
```

```
}
void
clearList()
{
    Node *bantu = head; while
    (bantu != NULL)
    {
        Node *hapus = bantu; bantu
        = bantu->next; delete
        hapus;
    } head = tail = NULL; cout << "List
berhasil terhapus!" << endl; }
void
tampil()
{ if (!isEmpty())
    {
        Node *bantu = head; while
        (bantu != NULL)
        { cout << bantu->data << " ";
            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;
}
```

Screenshots Output

```
3
3 5
2 3 5
1 2 3 5
2 3 5
2 3
2 7 3
2 3
1 3
1 8
1 11
PS C:\Users\Administrator\Documents\GitHub\Laporan-Praktikum-SDA\Modul3> 18102096 M.Fathiakmal.L.A
```

Deskripsi: Program di atas adalah sebuah program untuk mengelola sebuah Single linked list yang berisi bilangan bulat. Program ini dapat melakukan beberapa operasi dasar seperti menambahkan data di depan, belakang, atau di tengah linked list, menghapus data di depan, belakang, atau di tengah linked list, mengubah nilai data di depan, belakang, atau di tengah linked list, menampilkan isi linked list, dan menghapus semua data linked list. Data pada linked list ini terdiri dari sebuah struct Node yang berisi int data untuk menyimpan data bilangan bulat dan Node *next untuk menunjukkan ke data selanjutnya dalam linked list.

Guided 2

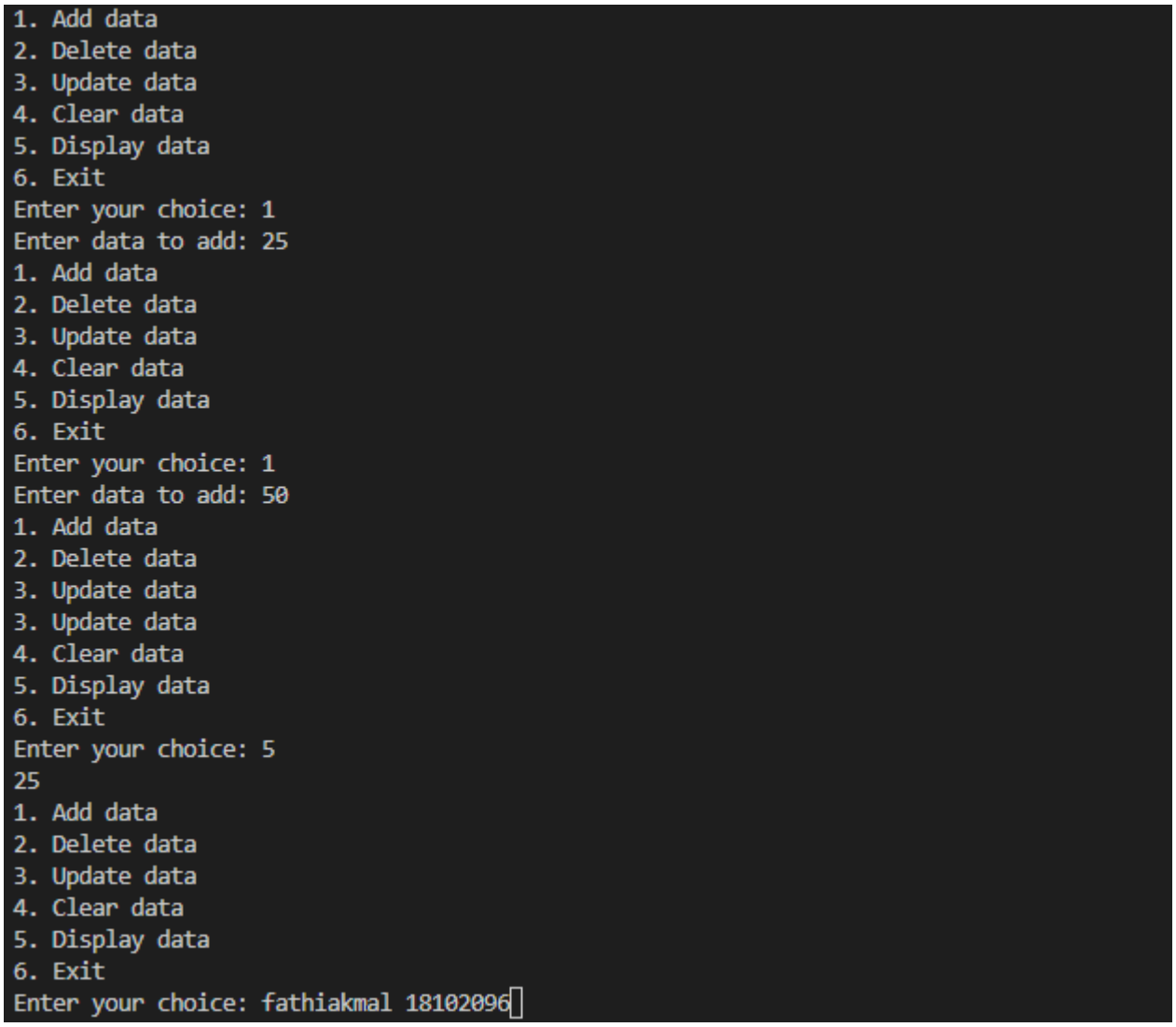
```
#include <iostream> using
namespace std;
class
Node
{
public:
    int data; Node
    *prev;
    Node *next;
};
class
DoublyLinkedList
{
public:
    Node *head;
    Node *tail;

    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;
}
```

Screenshots Output



Deskripsi: Program di atas adalah sebuah program yang mengimplementasikan Doubly Linked List menggunakan class. Program ini dapat melakukan beberapa operasi dasar pada Doubly Linked List, seperti menambahkan data, menghapus data, mengubah data, menghapus semua data, dan menampilkan data.

Didalam program terdapat 2 class yaitu:

- Node: class ini merepresentasikan simpul atau node dalam Doubly Linked List. Setiap node memiliki tiga anggota data yaitu int data untuk menyimpan nilai data, Node *prev untuk menunjukkan ke simpul sebelumnya, dan Node *next untuk menunjukkan ke simpul berikutnya.

- DoublyLinkedList: class ini berisi Doubly Linked List. Class ini memiliki dua anggota data yaitu Node *head yang menunjukkan ke simpul pertama dalam linked list, dan Node *tail yang menunjukkan ke simpul terakhir dalam linked list. Class ini juga dapat melakukan operasi pada linked list, seperti menambahkan data (push), menghapus data (pop), mengubah data (update), menghapus semua data (deleteAll), dan menampilkan data
(display).

Di dalam fungsi main(), program menampilkan menu untuk pengguna. Pengguna dapat memilih menu untuk menambahkan data baru, menghapus data, mengubah data, menghapus semua data, menampilkan data, atau keluar dari program. Program ini menggunakan switch-case untuk membuat menu dan menjalankan pilihan pengguna. Berikut adalah pilihan yang terdapat pada menu

- Add data: Menambahkan data baru ke depan Doubly Linked List.
- Delete data: Menghapus data dari depan Doubly Linked List.
- Update data: Mengubah data tertentu dalam Doubly Linked List.
- Clear data: Menghapus seluruh data dari Doubly Linked List.
- Display data: Menampilkan seluruh data dalam Doubly Linked List.
- Exit: Keluar dari program.

Program menggunakan perulangan do while yang tidak akan berhenti mengulang sampai mendapatkan inputan dari pengguna untuk keluar program.

C. Unguided/Tugas

Unguided 1

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

class Node {
public:

    string product_name;
    float price;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }
    void push(string product_name, float price) {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        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(string oldProduct, string newProduct, float newPrice)
    {
        Node* current = head;
        while (current != nullptr) {
            if (current->product_name == oldProduct) {
            current->product_name = newProduct;
            current->price = newPrice;
            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->product_name << " (Rp" << current->price
            << ")" << endl;
            current = current->next;
        }
    }

    void insert(string product_name, float price, int position) {
        if (position <= 0) {
        push(product_name, price);
        return;
    }
    Node* current = head;
    for (int i = 1; i < position && current != nullptr; i++) {
```

```
        current = current->next;
    }

    if (current == nullptr) {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        newNode->prev = tail;
        newNode->next = nullptr;
        if (tail != nullptr) {
            tail->next = newNode;
        } else {
            head = newNode;
        }
        tail = newNode;
    } else {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        newNode->prev = current->prev;
        newNode->next = current;
        if (current->prev != nullptr) {
            current->prev->next = newNode;
        } else {
            head = newNode;
        }
        current->prev = newNode;
    }
}

    void remove(int position) {
    if (position <= 0) {
        pop();
        return;
    }
    Node* current = head;
    for (int i = 1; i < position && current != nullptr; i++) {
        current = current->next;
    }
    if (current == nullptr) {
        return;
    }
    if (current == head) {
        head = current->next;
    } else {
        current->prev->next = current->next;
    }
```

```
    if (current == tail) {
        tail = current->prev;
    } else {
        current->next->prev = current->prev;
    }
    delete current;
    }
};

int main()
    {
    DoublyLinkedList list;
    int choice;
    string productName, newProductName;
    float price, newPrice;
    int position;
    do {
        cout << "1. tambah data\n";
        cout << "2. hapus data\n";
        cout << "3. Update data \n";
        cout << "4. tambah data urutan tertentu insert\n";
        cout << "5. hapus data urutan tertentu remove\n";
        cout << "6. hapus seluruh data\n";
        cout << "7. tampilkan data\n";
        cout << "8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "masukan nama produk: ";
            cin >> productName;
            cout << "masukan harga: ";
            cin >> price;
            list.push(productName, price);
            break;
        case 2:
            list.pop();
            break;
        case 3:
            cout << "masukan nama produk yang lama: ";
            cin >> productName;
            cout << "masukan nama produk yang baru: ";
            cin >> newProductName;
            cout << "masukan harga baru: ";
            cin >> newPrice;
            if (list.update(productName, newProductName, newPrice))
```

```
            {
            cout << "Produk berhasil ditambahkan\n";
            } else {
            cout << "produk tidak ada\n";
            }
            break;
        case 4:
            cout << "masukan nama produk: ";
            cin >> productName;
            cout << "masukan harga: ";
            cin >> price;
            cout << "masukan posisi: ";
            cin >> position;
            list.insert(productName, price, position);
            break;
        case 5:
            cout << "masukan urutan ke-: ";
            cin >> position;
            list.remove(position);
            break;
        case 6:
            list.deleteAll();
            cout << "semua produk berhasil di hapus\n";
            break;
        case 7:
            list.display();
            break;
        case 8:
            cout << "keluar dari progam...\n";
            break;
        default:
            cout << "pilihan salah\n";
            break;
        }
} while (choice != 8);

return 0;
}
```

Screenshots Output

```
# Menu Linked List Mahasiswa #

 1. Insert Depan
 2. Insert Belakang
 3. Insert Tengah
 4. Hapus Depan
 5. Hapus Belakang
 2. Insert Belakang
 3. Insert Tengah
 4. Hapus Depan
 5. Hapus Belakang
 6. Hapus Tengah
 7. Ubah Depan
 8. Ubah Belakang
 9. Ubah Tengah
 10. Tampilkan
 0. Keluar Program
 Pilihan : 1
Masukkan Nama : Fathi
Masukkan Usia : 24
 6. Hapus Tengah
 7. Ubah Depan
 8. Ubah Belakang
 9. Ubah Tengah
 10. Tampilkan
 0. Keluar Program
 Pilihan : 7
Masukkan Nama : Fathi
Masukkan Usia : 24

Fathi 24 , Fathiakmal 24 ,

 1. Insert Depan
 2. Insert Belakang
 3. Insert Tengah
```

```
4. Hapus Depan
5. Hapus Belakang
6. Hapus Tengah
7. Ubah Depan
8. Ubah Belakang
9. Ubah Tengah
10. Tampilkan
0. Keluar Program
Pilihan : Fathiakmal 182096
```

Deskripsi: Program di atas adalah implementasi linked list sederhana dalam C++ yang mendukung berbagai operasi seperti penambahan, penghapusan, pengubahan, dan penampilan data mahasiswa. Struktur Node digunakan untuk merepresentasikan setiap elemen dalam linked list, dengan masing-masing node menyimpan nama, usia, dan pointer ke node berikutnya. Fungsi init

menginisialisasi linked list, sementara fungsi isEmpty memeriksa apakah list kosong. Fungsi-fungsi seperti insertDepan, insertBelakang, dan insertTengah digunakan untuk menambah elemen di berbagai posisi, sedangkan hapusDepan, hapusBelakang, dan hapusTengah digunakan untuk menghapus elemen. Fungsi ubahDepan, ubahTengah, dan ubahBelakang mengubah nilai elemen yang ada di posisi tertentu. Program ini juga memiliki fungsi clearList untuk menghapus semua elemen dalam list dan tampil untuk menampilkan seluruh isi list. Menu interaktif di dalam fungsi main memungkinkan pengguna untuk memilih operasi yang diinginkan secara berulang hingga mereka memilih untuk keluar dari program.

Unguided 2

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

class Node {
public:

    string product_name;
    float price;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }
    void push(string product_name, float price) {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        newNode->prev = nullptr;
        newNode->next = head;
        if (head != nullptr) {
        head->prev = newNode;
        } else {
        tail = newNode;
        }
        head = newNode;
    }
```

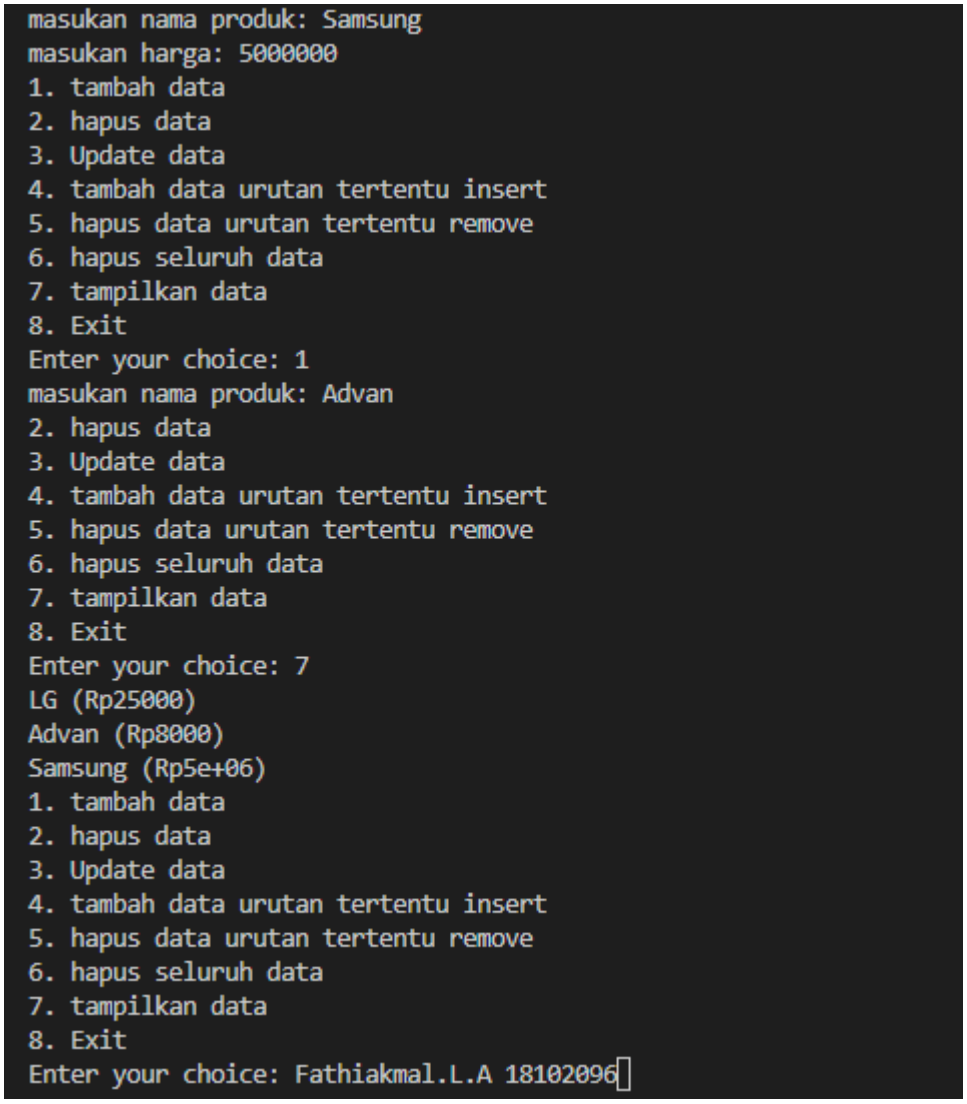
```
void pop() {
    if (head == nullptr) {
    return;
    }
    Node* temp = head;
    head = head->next;
    if (head != nullptr) {
    head->prev = nullptr;
    } else {
    tail = nullptr;
    }
    delete temp;
}

bool update(string oldProduct, string newProduct, float newPrice)
{
    Node* current = head;
    while (current != nullptr) {
        if (current->product_name == oldProduct) {
        current->product_name = newProduct;
        current->price = newPrice;
        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->product_name << " (Rp" << current->price
        << ")" << endl;
        current = current->next;
    }
```

```
    }

    void insert(string product_name, float price, int position) {
        if (position <= 0) {
        push(product_name, price);
        return;
    }
    Node* current = head;
    for (int i = 1; i < position && current != nullptr; i++) {
        current = current->next;
    }

    if (current == nullptr) {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        newNode->prev = tail;
        newNode->next = nullptr;
        if (tail != nullptr) {
            tail->next = newNode;
        } else {
            head = newNode;
        }
        tail = newNode;
    } else {
        Node* newNode = new Node;
        newNode->product_name = product_name;
        newNode->price = price;
        newNode->prev = current->prev;
        newNode->next = current;
        if (current->prev != nullptr) {
            current->prev->next = newNode;
        } else {
            head = newNode;
        }
        current->prev = newNode;
    }
}

    void remove(int position) {
    if (position <= 0) {
        pop();
        return;
    }
    Node* current = head;
    for (int i = 1; i < position && current != nullptr; i++) {
        current = current->next;
```

```
    }
    if (current == nullptr) {
        return;
    }
    if (current == head) {
        head = current->next;
    } else {
        current->prev->next = current->next;
    }
    if (current == tail) {
        tail = current->prev;
    } else {
        current->next->prev = current->prev;
    }
    delete current;
    }
};

int main()
    {
    DoublyLinkedList list;
    int choice;
    string productName, newProductName;
    float price, newPrice;
    int position;
    do {
        cout << "1. tambah data\n";
        cout << "2. hapus data\n";
        cout << "3. Update data \n";
        cout << "4. tambah data urutan tertentu insert\n";
        cout << "5. hapus data urutan tertentu remove\n";
        cout << "6. hapus seluruh data\n";
        cout << "7. tampilkan data\n";
        cout << "8. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
        case 1:
            cout << "masukan nama produk: ";
            cin >> productName;
            cout << "masukan harga: ";
            cin >> price;
            list.push(productName, price);
            break;
        case 2:
            list.pop();
```

```
            break;
        case 3:
            cout << "masukan nama produk yang lama: ";
            cin >> productName;
            cout << "masukan nama produk yang baru: ";
            cin >> newProductName;
            cout << "masukan harga baru: ";
            cin >> newPrice;
            if (list.update(productName, newProductName, newPrice))
            {
            cout << "Produk berhasil ditambahkan\n";
            } else {
            cout << "produk tidak ada\n";
            }
            break;
        case 4:
            cout << "masukan nama produk: ";
            cin >> productName;
            cout << "masukan harga: ";
            cin >> price;
            cout << "masukan posisi: ";
            cin >> position;
            list.insert(productName, price, position);
            break;
        case 5:
            cout << "masukan urutan ke-: ";
            cin >> position;
            list.remove(position);
            break;
        case 6:
            list.deleteAll();
            cout << "semua produk berhasil di hapus\n";
            break;
        case 7:
            list.display();
            break;
        case 8:
            cout << "keluar dari progam...\n";
            break;
        default:
            cout << "pilihan salah\n";
            break;
        }
} while (choice != 8);

return 0;
}
```

Screenshots Output



Program di atas adalah implementasi dari Doubly Linked List (DLL) dalam C++ untuk menyimpan dan mengelola data produk dengan nama dan harga. Kelas Node mendefinisikan struktur node dalam DLL dengan atribut product_name, price, prev, dan next. Kelas DoublyLinkedList mengelola operasi pada DLL, termasuk menambahkan node di depan (push), menghapus node dari depan (pop), memperbarui data node (update), menambah node di posisi tertentu (insert), menghapus node di posisi tertentu (remove), menghapus semua node (deleteAll), dan menampilkan seluruh node (display). Fungsi main menyediakan antarmuka pengguna berbasis teks untuk memilih berbagai operasi seperti menambah, menghapus, memperbarui, atau menampilkan data produk dalam DLL melalui menu interaktif hingga pengguna memilih untuk keluar dari program.

D. Kesimpulan

Single Linked List adalah struktur data yang terdiri dari serangkaian node yang terhubung satu sama lain melalui pointer 'next', dimana node pertama disebut head dan node terakhir menunjuk ke NULL. Sedangkan Doubly Linked List adalah varian dari linked list yang memiliki dua pointer penunjuk, yaitu ke node sebelumnya ('previous/prev') dan ke node berikutnya ('next'). Doubly Linked List memungkinkan traversing maju dan mundur, memudahkan operasi seperti penghapusan atau penambahan node di antara dua node yang ada. Di kedua jenis linked list ini, HEAD menunjuk pada node pertama dan TAIL menunjuk pada node terakhir, dengan nilai NULL menandakan linked list kosong. Perbedaan utama antara keduanya adalah kemampuan Doubly Linked List untuk bergerak maju dan mundur, sementara Single Linked List hanya bergerak maju.

E. Referensi

Mikirin Kode. (n.d.). Single Linked List. Diakses pada 29 Maret 2024, dari https://mikirinkode.com/single-linked-list/

Sekolah Ilmu Komputer (SICS) Binus University. (2017, 15 Maret). Doubly Linked List. Diakses pada 29 Maret 2024, dari https://socs.binus.ac.id/2017/03/15/doubly-linked-list/