LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA MODUL 3
"Linked List Dan Double Linked List"

DISUSUN OLEH:
Farrell Edric Kelvianto
2311102079
S1 IF-11-B

DOSEN:

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

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
INSTITUT TEKNOLOGI TELKOM PURWOKERTO
PURWOKERTO

2024

A. Dasar Teori

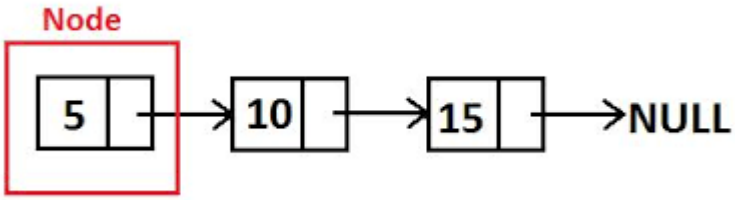
Linked List merupakan serangkaian elemen data yang mirip dengan array hanya saja dia disusun menggunakan urutan linear. Setiap elemen disebut Node. Linked listini sering juga disebut senarai berantai dan linked list ini bisa dianalogikan dengan rantai besi yang terdiri dari beberapa besi bulatan dan saling terhubung atau juga bisa dianalogikan dengan kereta yang memiliki sebuah banyak gerbong tanpa terputus yang disebut dengan pointer.

Sebagai berikut jenis linked list yang akan dibahas pada laporan praktikum ini:

1. Single Linked List

   Single Linked List ini merupakan linked list paling sederhana dan dirangkai bersama dengan cara sekuensial, atau sering disebut juga linked list linear. Setiap simpul itu dibagi menjadi 2 bagian yaitu satu bagian isi dan satu bagian pointer,untuk bagian isi itu berisi sebuah data yang disimpan oleh simpul. Bagian pointer, yaitu next yang berisikan alamat yang menunjuk ke setiap Node/simpul berikutnya. Untuk membuat Linked list ini dapat menggunakan 2 metode yaitu

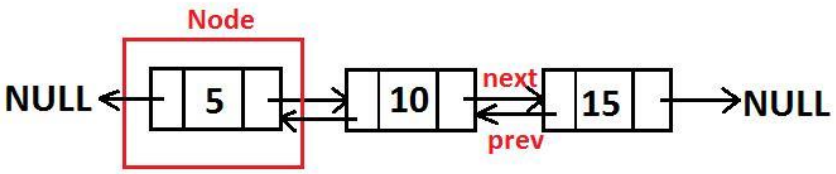
   a) LIFO (Last In First Out),Menggunakan aplikasi stack.
   b) FIFO (First In First Out) Menggunakan aplikasi antrian.



   Gambar diatas merupakan gambaran atau ilustrasi dari single linked list. Yang berwarna merah dengan kotak merah itu disebut head dan untuk tailnya itu berada di elemen akhir yaitu NULL. Kelemahan dari single linked list ini hanya dapat bergerak searah saja yaitu maju/mundur saja atau kanan/kiri sehingga untuk pencariannya hanya dapat bergerak searah saja.

2. Doubly Linked List

   Doubly linked list ini adalah struktur data Linked List yang mirip dengan single Linked List hanya saja ada tambahan dengan satu pointer pada setiap simpul yaitu pointer prev yang mengarah ke sebelumnya.



Guided

Guided 1

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

// Deklarasi Struct Node
struct Node {
    int data;
    Node* next;
};

Node* head;
Node* tail;

// Inisialisasi Node
void init() {
    head = NULL;
    tail = NULL;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == NULL;
}
```

```
// Tambah Node di depan
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;
    }
}

// Tambah Node di belakang
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;
    }
}

// Hitung jumlah Node di list
int hitungList() {
    Node* hitung = head;
    int jumlah = 0;
    while (hitung != NULL) {
        jumlah++;
        hitung = hitung->next;
    }
    return jumlah;
}

// Tambah Node di posisi 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 {
```

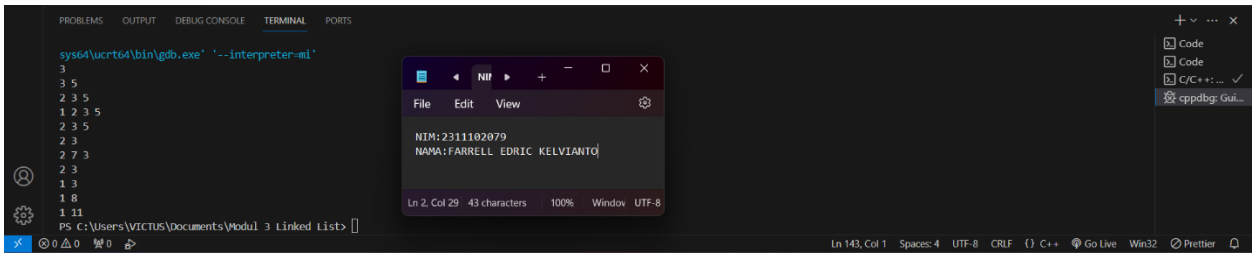
```
        Node* baru = new Node();
        baru->data = data;
        Node* bantu = head;
        int nomor = 1;
        while (nomor < posisi - 1) {
            bantu = bantu->next;
            nomor++;
        }
        baru->next = bantu->next;
        bantu->next = baru;
    }
}

// Hapus Node di depan
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;
    }
}

// Hapus Node di belakang
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;
    }
}

// Hapus Node di posisi tengah
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;
    }
}

// Ubah data Node di depan
void ubahDepan(int data) {
    if (!isEmpty()) {
        head->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Ubah data Node di posisi tengah
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;
    }
}

// Ubah data Node di belakang
void ubahBelakang(int data) {
    if (!isEmpty()) {
        tail->data = data;
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus semua Node di list
void clearList() {
    Node* bantu = head;
    while (bantu != NULL) {
        Node* hapus = bantu;
        bantu = bantu->next;
        delete hapus;
    }
    head = tail = NULL;
    cout << "List berhasil terhapus!" << endl;
}

// Tampilkan semua data Node di list
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;
}
```

Screenshot Output



Deskripsi Program

Cara kerja program ini adalah dimulai dari menu Init diawal program itu untuk mengatur sebuah pointer yang terdiri dari head dan tail,lanjut ada isEmpty itu untuk mengembalikan fungsi nilai yang jika linked list kosong maka linked list kosong dan jika tidak maka berisi,untuk insert depan fungsinya untuk menambahkan node baru didepan,insertBelakang berfungsi untuk menambahkan node dibelakang linked list,hitungList ini untuk menghitung sebuah jumlah Node didalam linked list,InsertTengah berfungsi untuk menambahkan nilai atau node baru pada posisi tertentu,hapusDepan ini berfungsi menghapus node pertama,hapusBelakang ini berfungsi untuk menghapus node terakhir,hapus tengah menghapus posisi tertentu,ubahDepan berfungsi untuk mengubah suatu nilai data dari node pertama,ubahTengah berfungsi untuk mengubah nilai data dari node pada posisi tertentu,ubahBelakang berfungsi untuk mengubah hanya node terakhir,clearList ini berfungsi menghapus semua node, dan terakhir berfungsi menampilkan sebuah node yang berada didalam linked list.

Guided 2

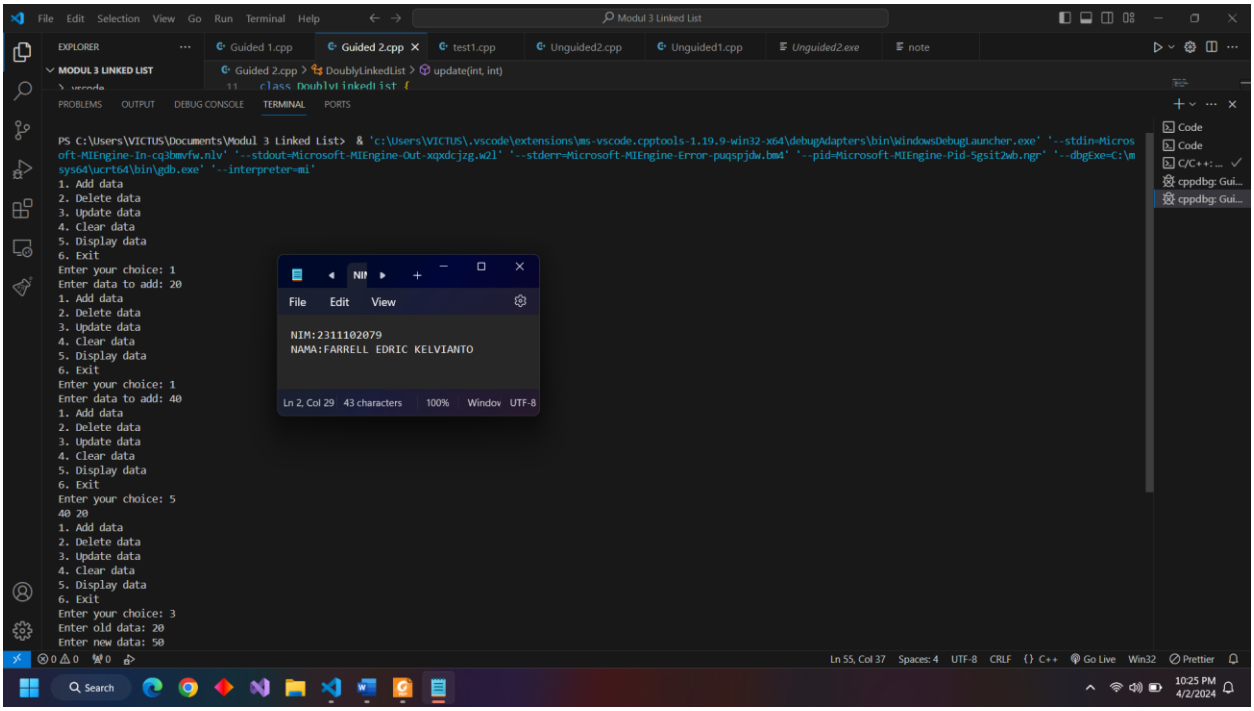
```
#include <iostream>
using namespace std;

class Node {
public:
    int data;
    Node* prev;
    Node* next;
```

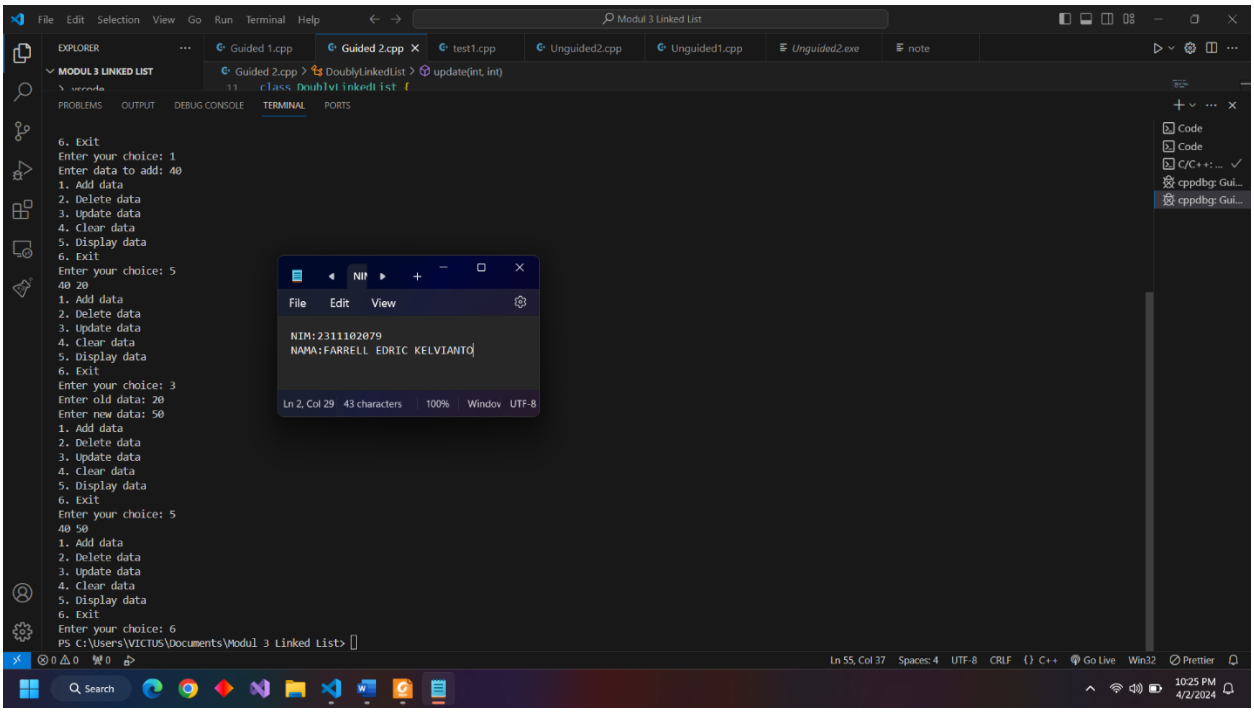
```
};

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;
}
```

Screenshot Output





Deskripsi Program

Untuk program kali ini menggunakan Double linked list yang lebih flexible dibandingkan Guided 1. Yang pertama ini ada class Node ini untuk mendefinisikan next dan prev yang berguna untuk rantainya di dalam Linked List dan sebagai penggeraknya. Di dalam program ini cukup singkat untuk membuat perintahnya yang pertama ada push dia sebagai membuat node baru di depan,menu selanjutnya adalah pop dia berfungsi menghapus node pertama dalam linked list,update ini untuk memperbarui nilai data tertentu dalam linked list,deleteAll ini berfungsi untuk menghapus semua elemen dari linked listnya dan terakhir adalah display untuk menampilkan semua elemen dari linked list.

B. Tugas

Unguided 1

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

// Struct
struct node {
    string iNPName;
    int Age;
    node* next;
};

node* head;
node* tail;

void inisial (){
    head = NULL;
    tail = NULL;
}

bool kosong(){
    return head == NULL;
}

void insert_front(string nama, int Age){
    node* baru = new node;
    baru->iNPName = nama;
    baru->Age = Age;
    baru->next = NULL;

    if (kosong()){
        head = tail = baru;
    } else {
        baru->next = head;
```

```
        head = baru;
    }
}

void insert_back(string iNPName, int Age){
    node* baru = new node;
    baru->iNPName = iNPName;
    baru->Age = Age;
    baru->next = NULL;

    if (head == NULL){
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

void insertCenter(string iNPName, int Age, int posisi){
    node* baru = new node;
    baru->iNPName = iNPName;
    baru->Age = Age;
    baru->next = NULL;

    if (head == NULL) {
        head = baru;
        tail = baru;
    } else {
        node* temp = head;
        int count = 1;

        while (temp != NULL && count < posisi - 1) {
            temp = temp->next;
            count++;
        }

        if (temp == NULL || posisi <= 0) {
            cout << "Posisi tidak valid." << endl;
            delete baru;
            return;
        }

        baru->next = temp->next;
        temp->next = baru;
```

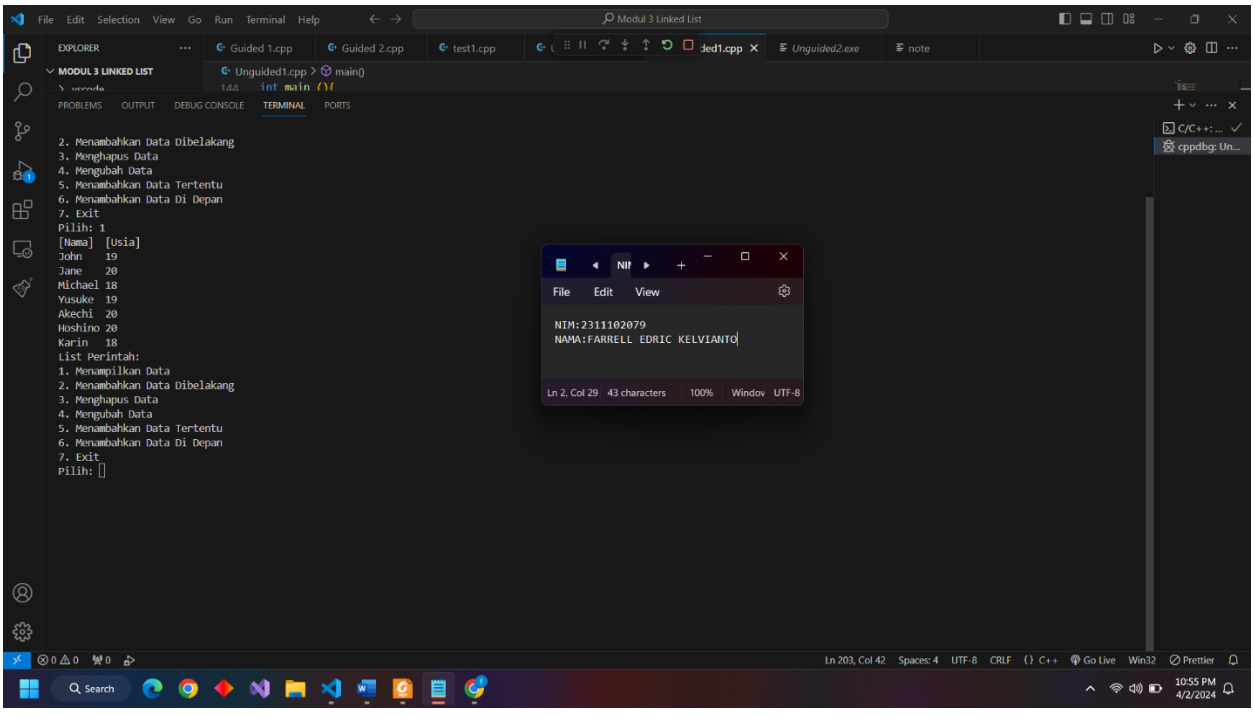
```
        if (baru->next == NULL) {
            tail = baru;
        }
    }
}

void Delete(string iNPName) {
    if (head == NULL) {
        cout << "Data kosong" << endl;
        return;
    }

    if (head->iNPName == iNPName) {
        node* hapus = head;
        head = head->next;
        delete hapus;
        if (head == NULL) {
            tail = NULL;
        }
        return;
    }

    node* temp = head;
    while (temp->next != NULL && temp->next->iNPName != iNPName) {
        temp = temp->next;
    }

    if (temp->next != NULL) {
        node* hapus = temp->next;
        temp->next = hapus->next;
        delete hapus;
        if (temp->next == NULL) {
            tail = temp;
        }
    } else {
        cout << "Data tidak ada" << endl;
        return;
    }
}

void changeData(string oldName, string newName, int newAge){
    node* temp = head;
    while (temp != NULL && temp->iNPName != oldName) {
        temp = temp->next;
    }
```

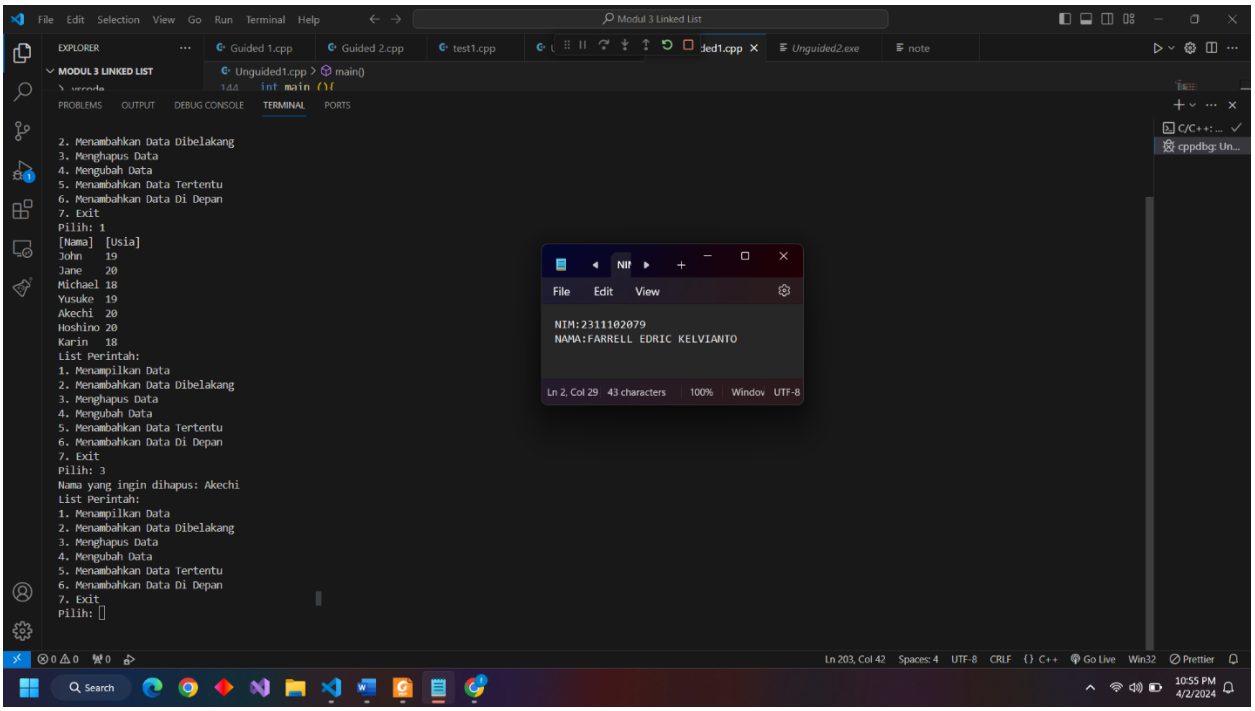
```
    if (temp != NULL){
        temp->iNPName = newName;
        temp->Age = newAge;
    } else {
        cout << "Data Tidak Ada" << endl;
    }
}

void Display (){
    if (head == NULL){
        cout << "List tidak ada" << endl;
        return;
    }
    node* temp = head;
    cout << "[Nama] " << "\t[Usia] " << endl;
    while (temp != NULL){
        cout << temp->iNPName << "\t" << temp->Age << endl;
        temp = temp->next;
    }
}

int main (){
    inisial();
    insert_front("John", 19);
    insert_back("Jane", 20);
    insert_back("Michael", 18);
    insert_back("Yusuke", 19);
    insert_back("Akechi", 20);
    insert_back("Hoshino", 20);
    insert_back("Karin", 18);

    int pilih;
    string nama, posisi, NewName;
    int usia, UsiaBaru;

    while (true){
        cout << "List Perintah: \n";
        cout << "1. Menampilkan Data" << endl;
        cout << "2. Menambahkan Data Dibelakang" << endl;
        cout << "3. Menghapus Data" << endl;
        cout << "4. Mengubah Data" << endl;
        cout << "5. Menambahkan Data Tertentu" << endl;
        cout << "6. Menambahkan Data Di Depan" << endl;
        cout << "7. Exit" << endl;
        cout << "Pilih: ";
```

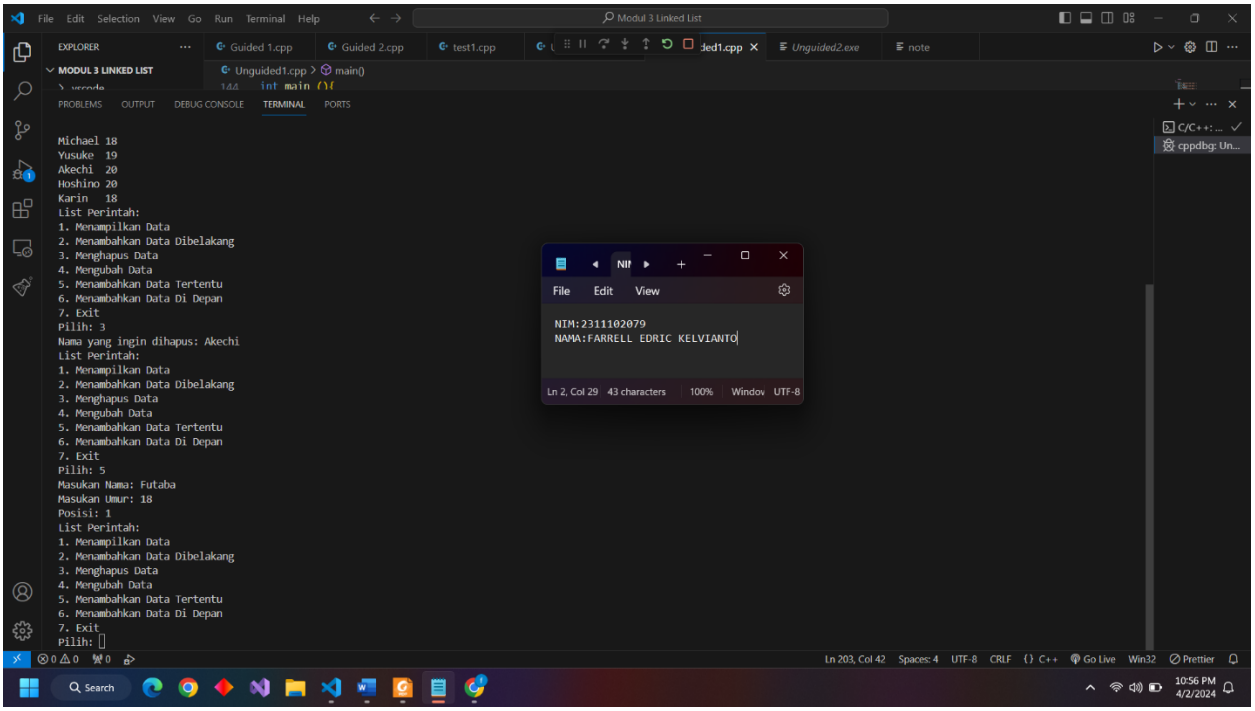
```
        cin >> pilih;

        switch (pilih){
            case 1:
                Display();
                break;
            case 2: {
                cout << "Masukan Nama: ";
                cin >> nama;
                cout << "Masukan Usia: ";
                cin >> usia;
                insert_back(nama, usia);
                break;
            }
            case 3: {
                cout << "Nama yang ingin dihapus: ";
                cin >> nama;
                Delete(nama);
                break;
            }
            case 4:{
                cout << "Masukan nama lama: ";
                cin >> nama;
                cout << "Masukan nama baru: ";
                cin >> NewName;
                cout << "Masukan Usia Baru: ";
                cin >> UsiaBaru;
                changeData(nama, NewName, UsiaBaru);
                break;
            }
            case 5:{
                string Name;
                int umur, posisi;
                cout << "Masukan Nama: ";
                cin >> Name;
                cout << "Masukan Umur: ";
                cin >> umur;
                cout << "Posisi: ";
                cin >> posisi;
                insertCenter(Name, umur, posisi);
                break;
            }
            case 6:
            {
                string nM;
```

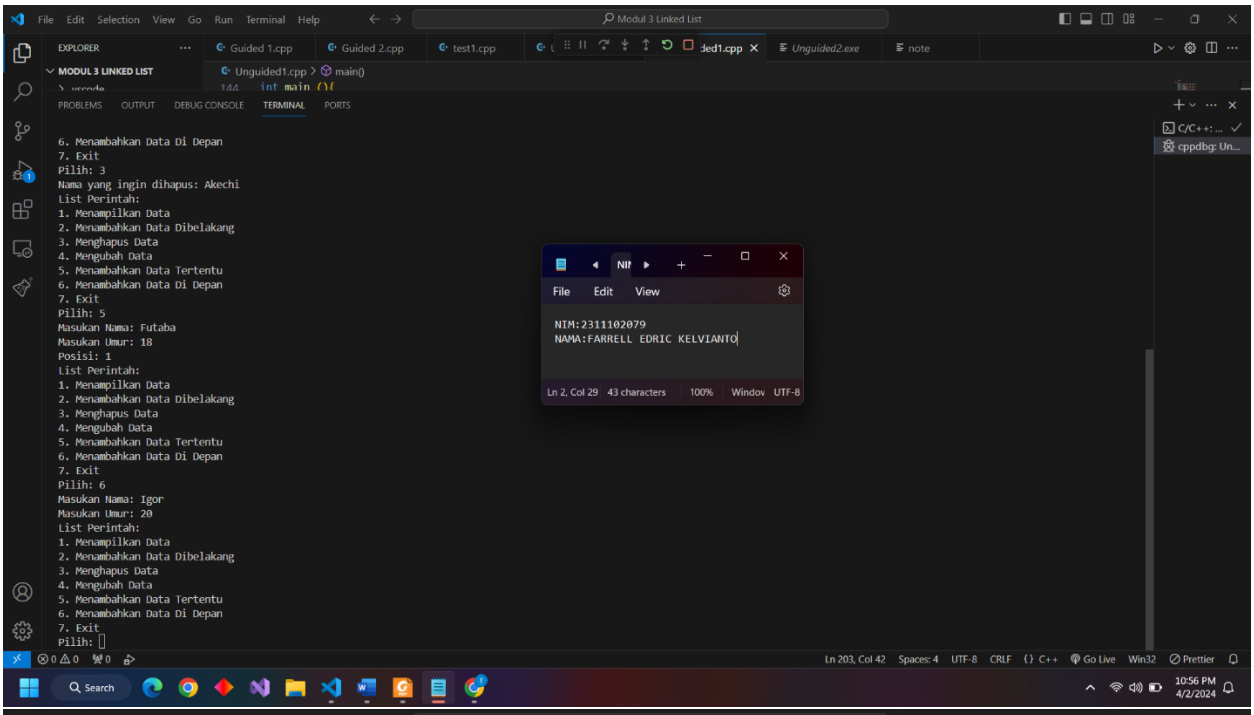
```
                int uMr;
                cout << "Masukan Nama: ";
                cin >> nM;
                cout << "Masukan Umur: ";
                cin >> uMr;
                insert_front(nM, uMr);
                break;
            }
            case 7:
                return 0;
            default:
                cout << "Anda salah input " << endl;
                break;
        }
    }
}
```

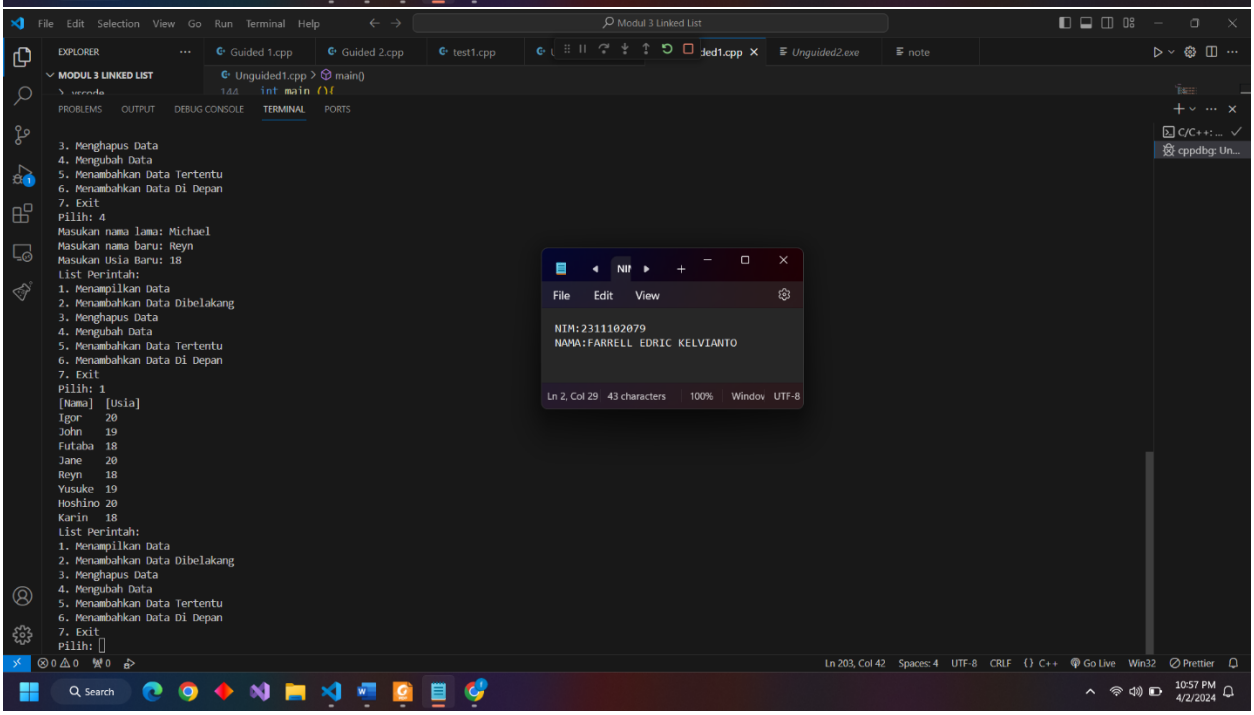
Screenshot Output




2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih: 1
[Nama] [Usia]
John 19
Jane 20
Michael 18
Yusuke 19
Akechi 20
Hoshino 20
Karin 18
List Perintah:
1. Menampilkan Data
2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih: 3
Nama yang ingin dihapus: Akechi
List Perintah:
1. Menampilkan Data
2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih:
NIM:2311102079
NAMA:FARRELL EDRIC KELVIANTO


Michael 18
Yusuke 19
Akechi 20
Hoshino 20
Karin 18
List Perintah:
1. Menampilkan Data
2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih: 3
Nama yang ingin dihapus: Akechi
List Perintah:
1. Menampilkan Data
2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih: 5
Masukan Nama: Futaba
Masukan Umur: 18
Posisi: 1
List Perintah:
1. Menampilkan Data
2. Menambahkan Data Dibelakang
3. Menghapus Data
4. Mengubah Data
5. Menambahkan Data Tertentu
6. Menambahkan Data Di Depan
7. Exit
Pilih:
NIM:2311102079
NAMA:FARRELL EDRIC KELVIANTO





Deskripsi Program

Dalam unguided pertama ini menggunakan single linked list dan untuk fungsi fungsinya sebagai berikut:

- Inisial( ) ini untuk menginisialisasi head dan tail menjadi null karena untuk data awal pasti masih kosong sehingga untuk deklarasi awal adalah null.
- Kosong ( ) berfungsi untuk mengembalikan nilai true jika linked list nya itu adalah kosong.
- Insertfront( ) berfungsi menerima inputan baru untuk data nama dan umurnya akan dimasukan ke node baru di depan.
- Insertcenter( ) berfungsi untuk mengubah elemen tertentu dalam linked list.
- Insertback( ) ini berfungsi untuk menambahkan nama dan umur di node baru bagian belakang.

- Delete ( ) berfungsi untuk menghapus bagian node dan nama tertentu.
- Changedata( ) berfungsi untuk mengubah data bagian tertentu
- Terakhir adalah display ( ) berfungsi untuk menampilkan seluruh inputan linked list dari user. Karena dari soal adalah yang diminta membuat data pertama yang dimasukan adalah nama dan usia kita dapat memakai perintah insertfront untuk menambahkannya itu untuk headnya untuk tailnya kita dapat menggunakan insertback untuk menambahkan dibelakang node head nya tadi. Lalu untuk selanjutnya diminta menghapus data akechi kita dapat menghapusnya menggunakan perintah delete dan ketikan Akechi untuk menghapusnya. Untuk menambahkan Futaba diantara john dan jane menggunakan insertcenter tinggal menginputkan nama dan umur lalu inputkan posisi diantara john dan jane. Selanjutnya untuk menambahkan igor di awal hanya menggunakan insertfront. Untuk mengubah datanya menggunakan prosedur changeData dan inputkan nama lama dan kemudian user akan menginputkan nama baru dan umur baru. Untuk menampilkan data hanya menggunakan display saja.

Unguided 2

```cpp
#include <iostream>
#include <iomanip>

using namespace std;

class Node {
public:
    string produk;
    int daftarHarga;
    Node* prev;
    Node* next;
};

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

    void pushFront(string produk, int daftarHarga) {
        Node* baru = new Node;
        baru->produk = produk;
        baru->daftarHarga = daftarHarga;
        baru->prev = NULL;
        baru->next = head;

        if (head != NULL) {
            head->prev = baru;
        }
```

```
        else {
            tail = baru;
        }
        head = baru;
    }

    void pushAtPosition(string produk, int daftarHarga, int pos) {
        Node* temp = head;
        int count = 1;

        while (temp != NULL && count < pos) {
            temp = temp->next;
            count++;
        }

        if (temp == NULL) {
            cout << "Posisi melebihi panjang list." << endl;
            return;
        }

        Node* baru = new Node;
        baru->produk = produk;
        baru->daftarHarga = daftarHarga;

        baru->prev = temp;
        baru->next = temp->next;

        if (temp->next != NULL) {
            temp->next->prev = baru;
        }
        else {
            tail = baru;
        }
        temp->next = baru;
    }

    void DeleteFront() {
     if (head == NULL) {
        cout << "Tidak ada List" << endl;
        return;
     }

     Node* temp = head;

     head = head->next;
```

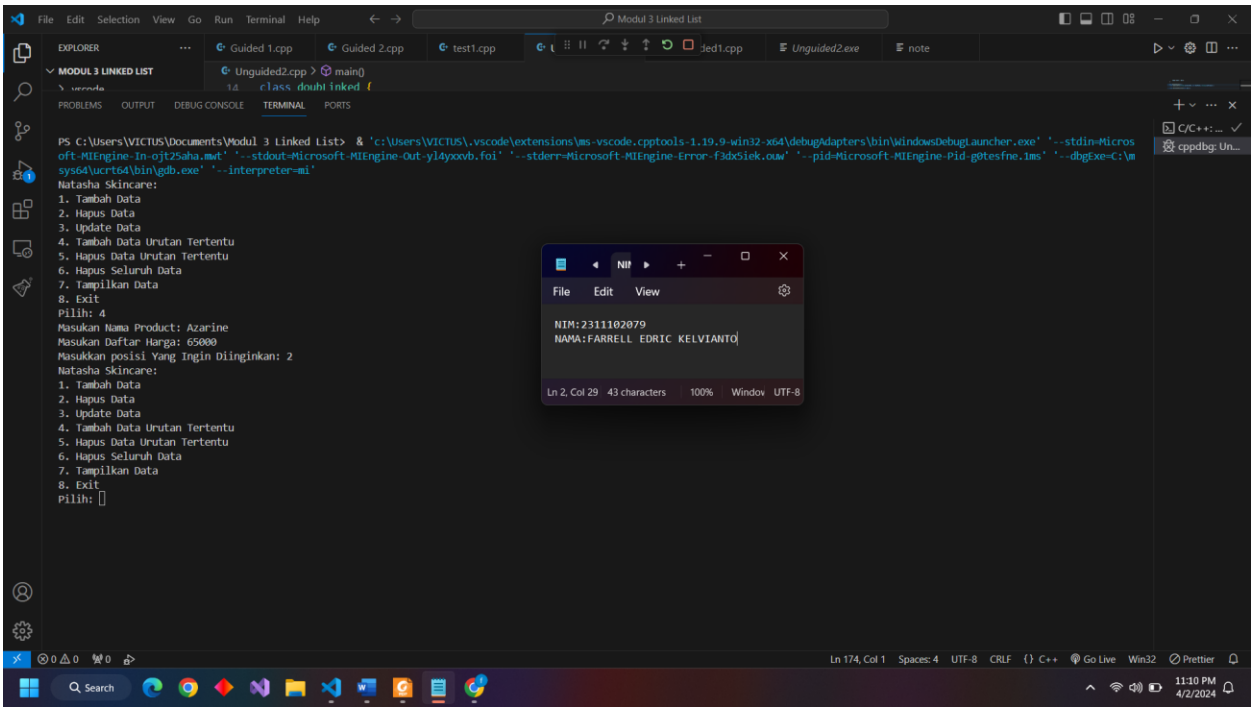
```
    if (head != NULL) {
        head->prev = NULL;
    } else {
        tail = NULL;
    }

    delete temp;
    cout << "Node Telah Dihapus" << endl;
}
    void UpdateDATA(string produkLAMA, int NewPRICE, string newProduk) {
        Node* temp = head;
        while (temp != NULL && temp->produk != produkLAMA) {
            temp = temp->next;
        }
        if (temp == NULL) {
            cout << "Produk tidak ada." << endl;
            return;
        }
        temp->produk = newProduk;
        temp->daftarHarga = NewPRICE;
    }

    void DeleteAtPosition(int pos) {
        Node* temp = head;
        int count = 1;

        while (temp != NULL && count < pos) {
            temp = temp->next;
            count++;
        }

        if (temp == NULL || temp->next == NULL) {
            cout << "Posisi melebihi panjang list." << endl;
            return;
        }

        Node* toDelete = temp->next;
        temp->next = toDelete->next;

        if (toDelete->next != NULL) {
            toDelete->next->prev = temp;
        }
        else {
            tail = temp;
        }
```

```
        delete toDelete;
    }

    void DeleteAll() {
        Node* current = head;
        while (current != NULL) {
            Node* nextNode = current->next;
            delete current;
            current = nextNode;
        }
        head = NULL;
        tail = NULL;
    }

    void Display() {
        Node* temp = head;

        cout << "+------------------------+----------+" << endl;
        cout << "|       Nama Produk      |  Harga   |" << endl;
        cout << "+------------------------+----------+" << endl;

        while (temp != NULL) {
            cout << "| " << left << setw(23) << temp->produk << "| " << right
<< setw(8) << temp->daftarHarga << " |" << endl;
            cout << "+------------------------+----------+" << endl; //
Tambahkan garis setiap kali memproses entri
            temp = temp->next;
        }
    }
};

int main() {
    int pilih;
    doubLinked list;

    list.pushFront("Hanasui", 60000);
    list.pushFront("Wardah", 150000);
    list.pushFront("Skintific", 100000);
    list.pushFront("Somethinc", 50000);
    list.pushFront("Orginote", 30000);

    while (true) {
        cout << "Natasha Skincare: " << endl;
        cout << "1. Tambah Data" << endl;
        cout << "2. Hapus Data" << endl;
```

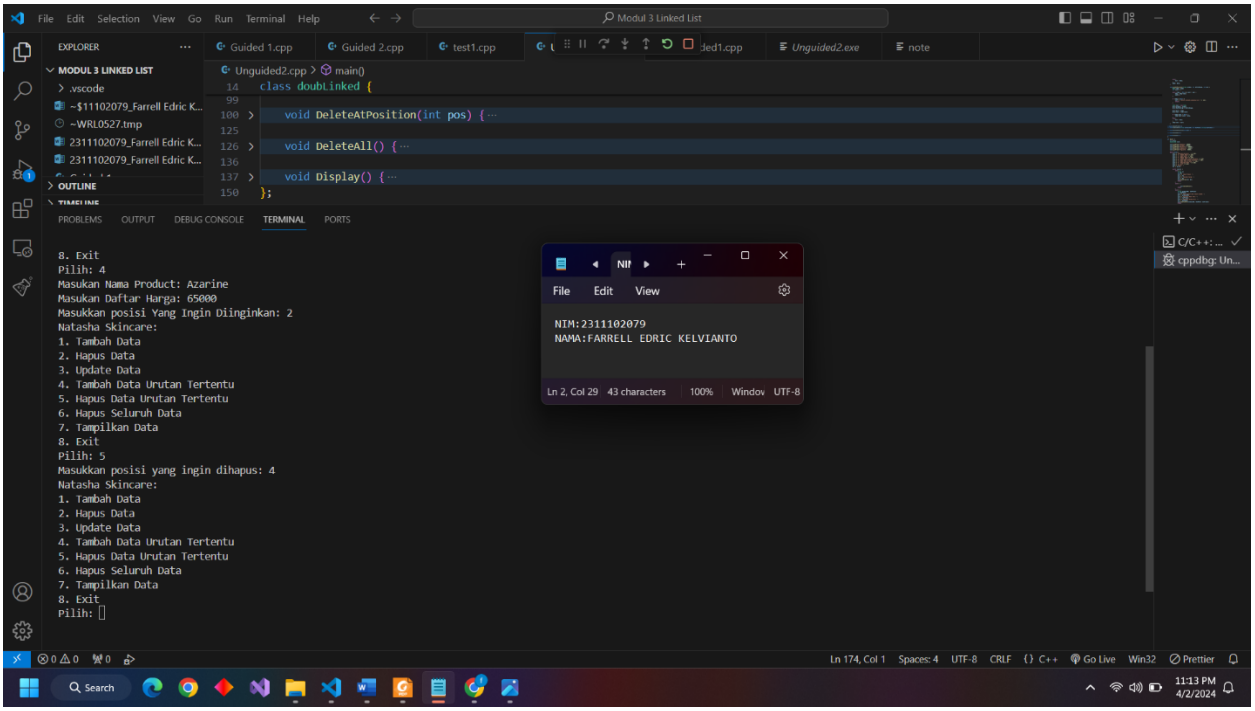
```
        cout << "3. Update Data" << endl;
        cout << "4. Tambah Data Urutan Tertentu" << endl;
        cout << "5. Hapus Data Urutan Tertentu" << endl;
        cout << "6. Hapus Seluruh Data" << endl;
        cout << "7. Tampilkan Data " << endl;
        cout << "8. Exit" << endl;
        cout << "Pilih: ";
        cin >> pilih;

        switch (pilih) {
            case 1: {
                string nP;
                int dH;
                cout << "Nama Produk: ";
                cin >> nP;
                cout << "Masukan Harga: ";
                cin >> dH;
                list.pushFront(nP, dH);
                break;
            }
            case 2: {

                    list.DeleteFront();
                break;
            }
            case 3: {
                string produkLAMA, newProduk;
                int NewPRICE;
                cout << "Nama Produk yang ingin diubah: ";
                cin >> produkLAMA;
                cout << "Masukkan Nama Baru: ";
                cin >> newProduk;
                cout << "Masukkan Harga Baru: ";
                cin >> NewPRICE;
                list.UpdateDATA(produkLAMA, NewPRICE, newProduk);
                break;
            }
            case 4: {
                int pos;
                cout << "Masukkan posisi Yang Ingin Dihapus: ";
                cin >> pos;
                list.pushAtPosition(pos);
                break;
            }
            case 5: {
```

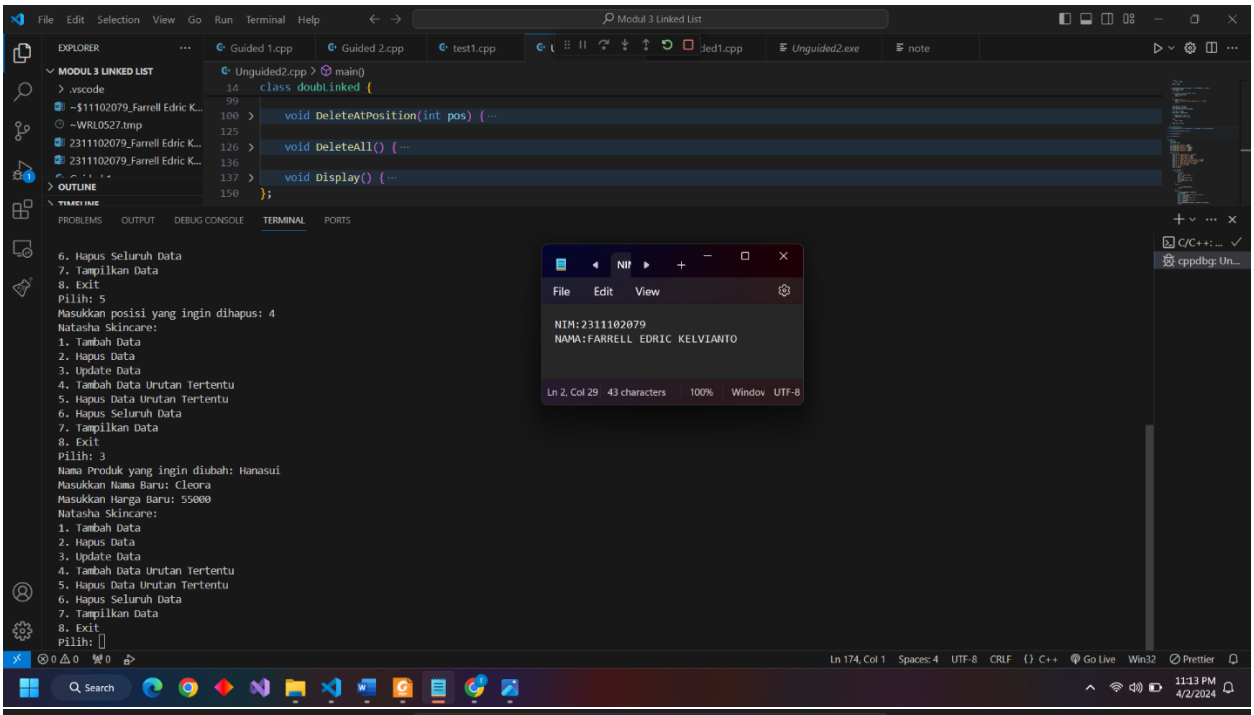
```
                int pos;
                cout << "Masukkan posisi yang ingin dihapus: ";
                cin >> pos;
                list.DeleteAtPosition(pos);
                break;
            }
            case 6: {
                list.DeleteAll();
                cout << "Semua data telah dihapus." << endl;
                break;
            }
            case 7: {
                list.Display();
                break;
            }
            case 8:
                cout << "Program selesai." << endl;
                return 0;
            default:
                cout << "Salah Input" << endl;
                break;
        }
    }

    return 0;
}
```

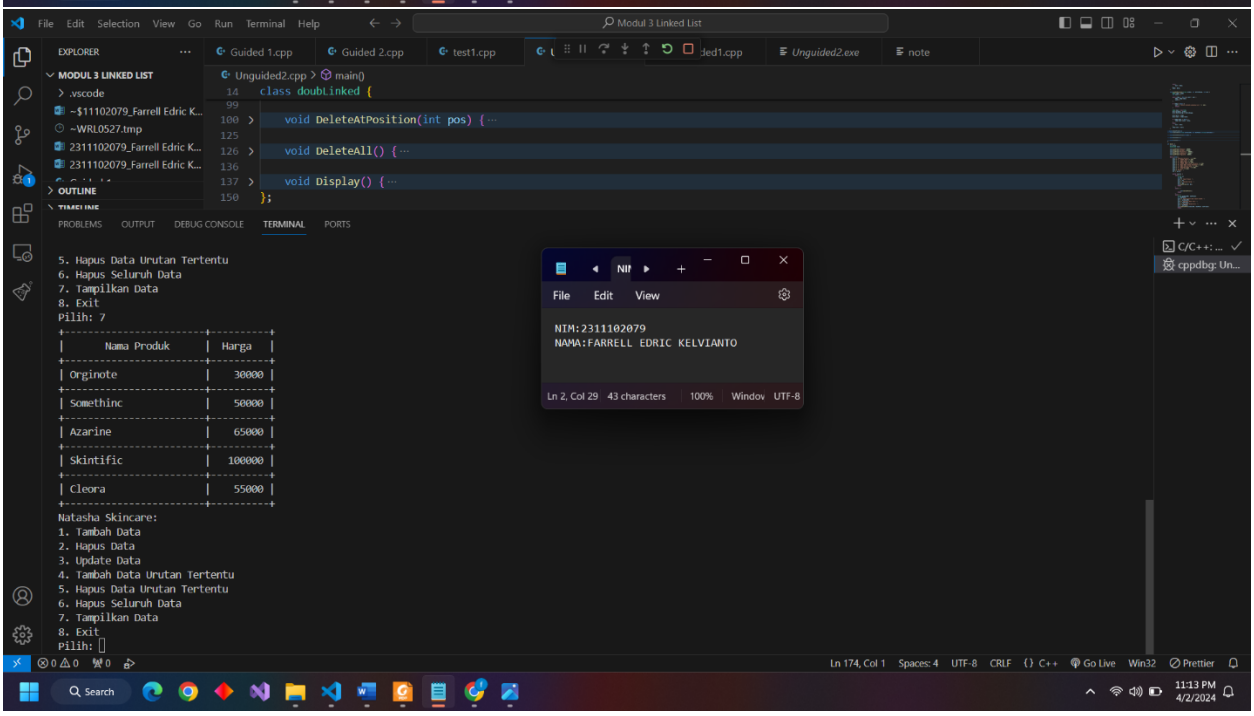
Screenshot Output


Modul 3 Linked List
Unguided2.cpp
PS C:\Users\VICTUS\Documents\Modul 3 Linked List> & 'c:\Users\VICTUS\.vscode\extensions\ms-vscode.cpptools-1.19.9-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-ojt25aha.mwt' '--stdout=Microsoft-MIEngine-Out-yl4yxxvb.foi' '--stderr=Microsoft-MIEngine-Error-f3dx5iek.ouw' '--pid=Microsoft-MIEngine-Pid-g0tesfne.1ms' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter=mi'
Natasha Skincare:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih: 4
Masukan Nama Product: Azarine
Masukan Daftar Harga: 65000
Masukkan posisi Yang Ingin Diinginkan: 2
Natasha Skincare:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih:
File Edit View
NIM:2311102079
NAMA:FARRELL EDRIC KELVIANTO
Ln 2, Col 29 43 characters 100% Windov UTF-8

Z


Modul 3 Linked List
Unguided2.cpp
class doubLinked {
void DeleteAtPosition(int pos) {
void DeleteAll() {
void Display() {
};
8. Exit
Pilih: 4
Masukan Nama Product: Azarine
Masukan Daftar Harga: 65000
Masukkan posisi Yang Ingin Diinginkan: 2
Natasha Skincare:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih: 5
Masukkan posisi yang ingin dihapus: 4
Natasha Skincare:
1. Tambah Data
2. Hapus Data
3. Update Data
4. Tambah Data Urutan Tertentu
5. Hapus Data Urutan Tertentu
6. Hapus Seluruh Data
7. Tampilkan Data
8. Exit
Pilih:
File Edit View
NIM:2311102079
NAMA:FARRELL EDRIC KELVIANTO
Ln 2, Col 29 43 characters 100% Windov UTF-8





Deskripsi Program

Berikut penjelasan masing – masing prosedur didalam doubly Linked List ini:

- Pushfront ( ) berfungsi untuk menambahkan node head baru .
- PushAtPosition ( ) ini berfungsi untuk menambahkan data tertentu yang berarti dapat milih dilist keberapa untuk ditambahkan.
- DeleteFront ( ) ini berfungsi untuk menghapus bagian kepala/head dari node tersebut.
- UpdateData( ) ini berfungsi untuk mengubah data lama menjadi data baru dengan cara user

menggantinya dengan cara menginputkan nama yang akan diubah lalu user akan diminta nama yang mau diubah beserta harganya.

- Deleteatposition ( ) ini berfungsi untuk menghapus linked list tertentu,dan untuk cara menghapusnya dia dimulai dari indeks ke 0 dan hanya mencari urutan keberapa untuk menghapusnya.
- DeleteAll( ) ini berfungsi menghapus semua list didalam linked list tersebut.
- Yang terakhir ada display ( ) ini untuk menampilkan linked list yang telah ada dan inputan dari user.

Karena dari soal yang diminta adalah menambahkan produk diantara somethinck dan skintific maka kita perlu yang namanya prosedur pushAtPosition( ), untuk menghapus Wardah kita dapat menggunakan prosedur DeleteAtPosition( ),untuk update produk Hanasui menjadi cleora dengan harga 55.000 kita dapat menggunakan prosedur updateData ( ), dan yang terakhir ada menu tersebut berada di screenshot output.

Kesimpulan

Kesimpulannya adalah linked list ini lebih dinamis dan lebih cepat juga untuk menambahkan dan mengurangkan pada data. Linked List dapat menyimpan dan mengelola data dalam urutan tertentu hampir mirip dengan array hanya saja dia lebih flexible untuk mengatur sebuah data. Linked List ini sering diimplementasikan seperti stack, queue dan graf.

C. Refrensi

Asisten Praktikum. “Modul 3 Linked List”. Learning Management System. 2024

DAFTAR PUSTAKA