LAPORAN PRAKTIKUM STRUKTUR DATA DAN ALGORITMA

MODUL 3

SINGLE AND DOUBLE LINKED LIST

DISUSUN OLEH:

PRIESTY AMEILIANA MAULIDAH

2311102175

S1 IF-11-E

DOSEN:

Muhammad Afrizal Amrustian, S. Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO PURWOKERTO

2024

A.DASAR TEORI

. Linked List adalah struktur data yang berisi kumpulan node yang terhubung secara sekuensial melalui pointer. Setiap node terdiri dari dua bagian, yaitu informasi dan pointer ke node berikutnya atau sebelumnya. Linked List dapat digunakan untuk operasi penambahan, penghapusan, dan pencarian nilai pada simpul tertentu. Single Linked List hanya memerlukan satu pointer untuk setiap simpul, sehingga lebih efisien dalam penggunaan memori dibandingkan dengan Linked List lainnya. Penggunaan memori yang lebih besar dan waktu eksekusi yang lebih lama terjadi pada Double Linked List, yang memiliki pointer prev untuk simpul sebelumnya. Namun, Double Linked List memungkinkan operasi penambahan dan penghapusan simpul yang lebih fleksibel dan traversal dari depan atau belakang dengan mudah.

B.Guided

Guided 1

```
// priesty ameiliana maulidah
// 2311102175


#include <iostream>
using namespace std;


///PROGRAM SINGLE LINKED LIST NON-CIRCULAR
//Deklarasi Struct Node
struct Node{
//komponen/member
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;
}
}
```

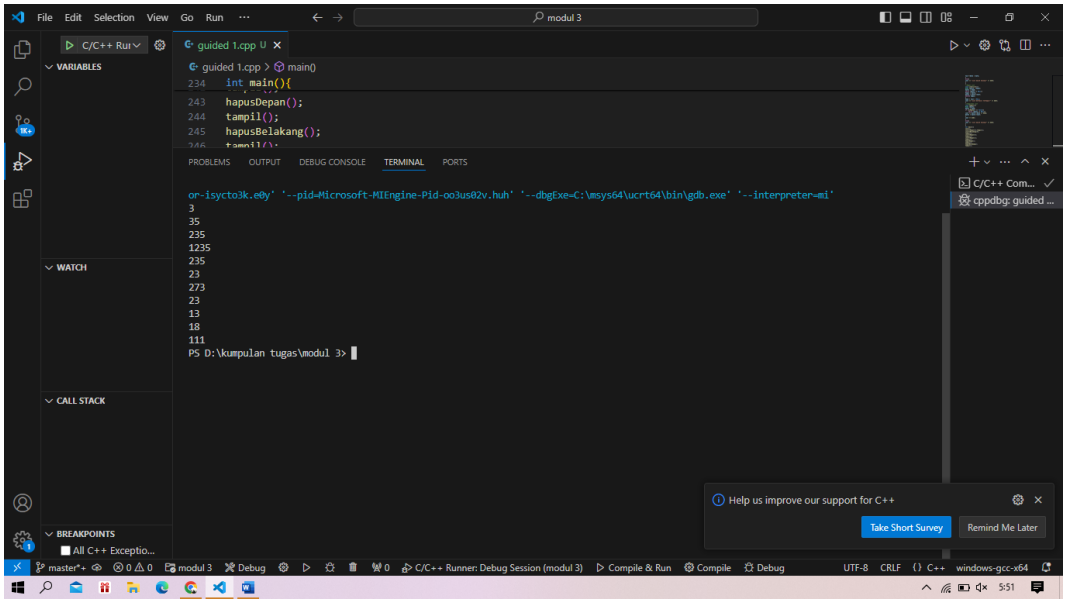
```
//Ubah Depan
void ubahDepan(int data){
if (isEmpty() == false){
head->data = data;
}
else{
cout << "List masih kosong!" << endl;
}
}
//Ubah Tengah
void ubahTengah(int data, int posisi){
Node *bantu;
if (isEmpty() == false){
if( posisi < 1 || posisi > hitungList() ){
cout << "Posisi di luar jangkauan" <<

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;
}
```

Screenshots output



Deskripsi :

Struktur Data:

- **Node:** Digunakan untuk menyimpan data dalam list. Setiap node memiliki dua komponen:
    - data: Menyimpan nilai data.
    - next: Pointer ke node berikutnya dalam list.
- **Head:** Pointer ke node pertama dalam list.
- **Tail:** Pointer ke node terakhir dalam list.

Fungsi-fungsi:

- init(): Inisialisasi list dengan setting head dan tail ke NULL.

- isEmpty(): Memeriksa apakah list kosong. Mengembalikan true jika list kosong, false jika tidak.
- insertDepan(int nilai): Menambahkan node baru dengan nilai yang diberikan di depan list.
- insertBelakang(int nilai): Menambahkan node baru dengan nilai yang diberikan di belakang list.
- hitungList(): Menghitung jumlah node dalam list.
- insertTengah(int data, int posisi): Menambahkan node baru dengan nilai dan posisi yang diberikan di tengah list.
- hapusDepan(): Menghapus node pertama dari list.
- hapusBelakang(): Menghapus node terakhir dari list.
- hapusTengah(int posisi): Menghapus node pada posisi yang diberikan dari list.
- ubahDepan(int data): Mengubah nilai data node pertama dalam list.
- ubahTengah(int data, int posisi): Mengubah nilai data node pada posisi yang diberikan dalam list.
- ubahBelakang(int data): Mengubah nilai data node terakhir dalam list.

- clearList(): Menghapus semua node dari list.
- tampil(): Menampilkan isi list.

Guided 2

```
// priesty ameiliana maulidah
// 2311102175


#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;
}
```

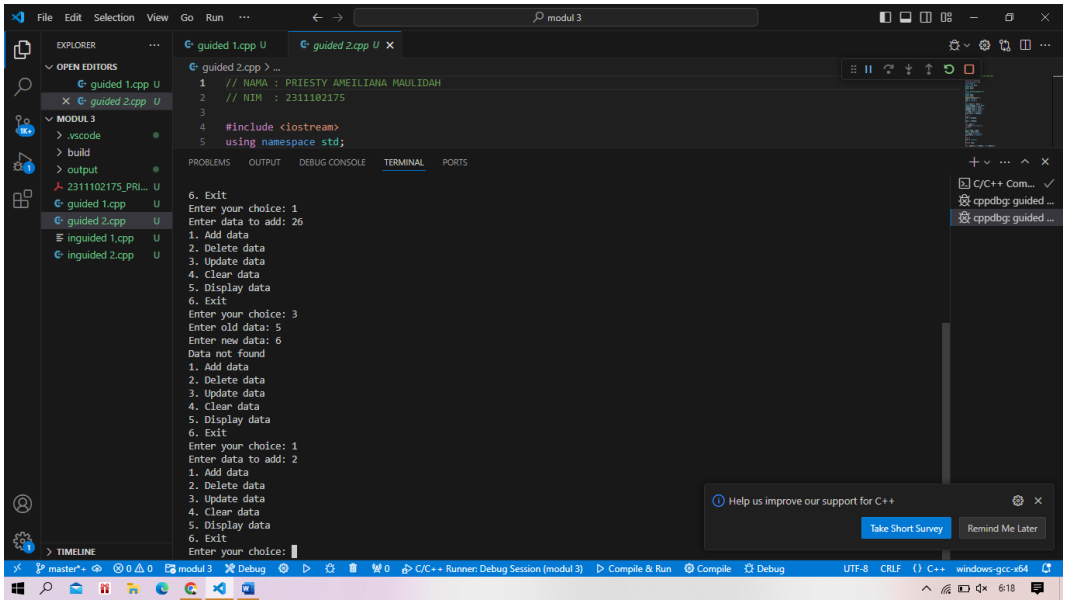
```
bool update(int oldData, int newData) {
Node* current = head;while (current != nullptr) {
if (current->data == oldData) {
current->data = newData;
return true;
}
current = current->next;
}
return false;
}
void deleteAll() {
Node* current = head;
while (current != nullptr) {
Node* temp = current;
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 :

- prev: Menunjuk ke node sebelumnya dalam list.
- next: Menunjuk ke node selanjutnya dalam list.

Program ini menyediakan class DoublyLinkedList yang memiliki beberapa fungsi untuk mengoperasikan list:

- **Konstruktor:** Meminisiasi head dan tail menjadi nullptr (menandakan list kosong).
- push(int data): Menambahkan node baru dengan nilai data di depan list.
- pop(): Menghapus node pertama dari list.
- update(int oldData, int newData): Mencari node dengan nilai oldData dan mengubahnya menjadi

newData. Mengembalikan true jika data ditemukan, false jika tidak ditemukan.

- deleteAll(): Menghapus semua node dari list.
- display(): Menampilkan isi list dari awal sampai akhir.

c. unguided/tugas

unguided 1

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>
#include <string>

using namespace std;

// Struct for Node
struct Node {
    string name;
    int age;
    Node *next;
};

Node *head;
Node *tail;

// Initialize the linked list
void init() {
    head = NULL;
    tail = NULL;
}
```

```
// Check if the list is empty
bool isEmpty() {
   return head == NULL;
}

// Insert at the front
void insertDepan(string name, int age) {
   Node *newNode = new Node;
   newNode->name = name;
   newNode->age = age;
   newNode->next = NULL;
   if (isEmpty()) {
      head = tail = newNode;
   } else {
      newNode->next = head;
      head = newNode;
   }
}
```

```
// Count the number of nodes in the list
int hitungList() {
   Node *current = head;
   int count = 0;
   while (current != NULL) {
      count++;
      current = current->next;
   }
   return count;
}

// Insert in the middle
void insertTengah(string name, int age, int posisi) {
   if (posisi < 1 || posisi > hitungList()) {
      cout << "Posisi di luar jangkauan" << endl;
   } else if (posisi == 1) {
      cout << "Posisi bukan posisi tengah" << endl;
   } else {
      Node *newNode = new Node;
      newNode->name = name;
      newNode->age = age;
      Node *current = head;
      int nomor = 1;
      while (nomor < posisi - 1) {
         current = current->next;
         nomor++;
      }
```

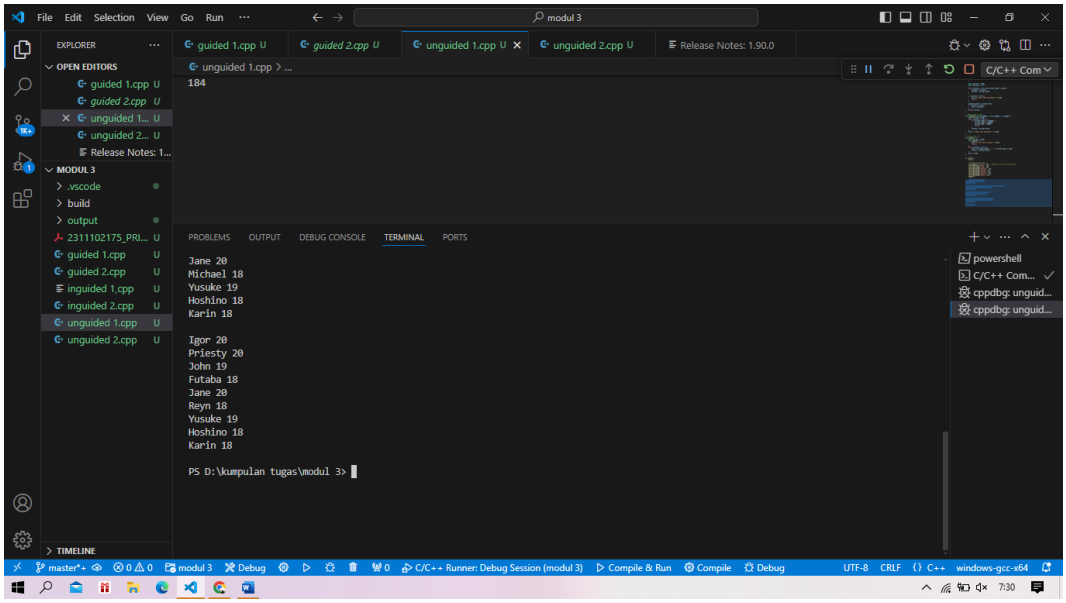
```
newNode->next = current->next;
        current->next = newNode;
    }
}

// Delete from the middle by name
void hapusTengah(string name) {
    if (isEmpty()) {
        cout << "List kosong!" << endl;
        return;
    }

    if (head->name == name) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        return;
    }
Node *current = head;
    Node *previous = NULL;

    while (current != NULL && current->name != name) {
        previous = current;
        current = current->next;
    }
```

```
if (current == NULL) {
        cout << "Nama tidak ditemukan!" << endl;
        return;
    }

    previous->next = current->next;
    if (current == tail) {
        tail = previous;
    }
    delete current;
}

// Update data by name
void ubahData(string oldName, string newName, int newAge) {
    Node *current = head;
    while (current != NULL) {
        if (current->name == oldName) {
            current->name = newName;
            current->age = newAge;
            return;
        }
        current = current->next;
    }
    cout << "Nama tidak ditemukan!" << endl;
}
```

```
// Display the list
void tampil() {
   Node *current = head;
   if (isEmpty()) {
      cout << "List masih kosong!" << endl;
      return;
   }
   while (current != NULL) {
      cout << current->name << " " << current->age << endl;
      current = current->next;
   }
   cout << endl;
}

int main() {
   init();

   // Insert initial data
   insertDepan("Priesty", 20); // Replace with your name and age
   insertBelakang("John", 19);
   insertBelakang("Jane", 20);
   insertBelakang("Michael", 18);
   insertBelakang("Yusuke", 19);
   insertBelakang("Akechi", 20);
   insertBelakang("Hoshino", 18);
   insertBelakang("Karin", 18);
   tampil();
```

```
// Hapus data Akechi
    hapusTengah("Akechi");
    tampil();

    // Tambahkan data Futaba diantara John dan Jane
    insertTengah("Futaba", 18, 3);
    tampil();

    // Tambahkan data Igor diawal
    insertDepan("Igor", 20);
    tampil();

    // Ubah data Michael menjadi Reyn
    ubahData("Michael", "Reyn", 18);
    tampil();

    return 0;
}
```

Screenshot output



Deskripsi :

- name (nama): untuk menyimpan nama seseorang (bisa diganti dengan data lain sesuai kebutuhan)
- age (umur): untuk menyimpan umur seseorang (bisa diganti dengan data lain sesuai kebutuhan)
- next (selanjutnya): pointer yang menunjuk ke node berikutnya di dalam linked list

Fungsi-Fungsi Program

Program ini memiliki beberapa fungsi untuk memanipulasi linked list:

- init(): Meminisiasi linked list dengan membuat head dan tail bernilai NULL, menandakan linked list masih kosong.

- isEmpty(): Mengecek apakah linked list kosong. Mengembalikan nilai true jika kosong, sebaliknya false.
- insertDepan(name, age): Menambahkan node baru berisi name dan age di depan linked list.
- insertBelakang(name, age): Menambahkan node baru berisi name dan age di belakang linked list.
- hitungList(): Menghitung jumlah node yang ada di dalam linked list.
- insertTengah(name, age, posisi): Menambahkan node baru berisi name dan age pada posisi tertentu di dalam linked list (posisi dimulai dari 1). Fungsi ini akan menampilkan pesan kesalahan jika posisi di luar jangkauan atau bukan posisi tengah.
- hapusTengah(name): Menghapus node yang memiliki name yang diberikan dari linked list. Fungsi ini akan menampilkan pesan kesalahan jika linked list kosong atau nama tidak ditemukan.
- ubahData(oldName, newName, newAge): Mengubah name dan age dari node yang memiliki oldName yang diberikan. Fungsi ini akan menampilkan pesan kesalahan jika nama tidak ditemukan.

- tampil(): Menampilkan seluruh isi linked list, yaitu name dan age dari setiap node.

Penggunaan Program

1. **Inisialisasi**: Program dimulai dengan menginisialisasi linked list menggunakan fungsi init().
2. **Memasukkan Data Awal**: Beberapa data awal berupa nama dan umur dimasukkan ke dalam linked list menggunakan fungsi insertDepan() dan insertBelakang().
3. **Menampilkan Isi**: Fungsi tampil() digunakan untuk menampilkan isi linked list saat ini.
4. **Operasi**: Program kemudian mendemonstrasikan berbagai operasi pada linked list, seperti:

- Menghapus data "Akechi" menggunakan hapusTengah("Akechi").
- Menambahkan data "Futaba" di antara "John" dan "Jane" menggunakan insertTengah("Futaba", 18, 3).
- Menambahkan data "Igor" di awal menggunakan insertDepan("Igor", 20).
- Mengubah data "Michael" menjadi "Reyn" menggunakan ubahData("Michael", "Reyn", 18).

5. **Penampilan Terakhir**: Setelah setiap operasi, fungsi tampil() dipanggil kembali untuk menunjukkan perubahan pada linked list.

Unguided2

```
// priesty ameiliana maulidah
// 2311102175

#include <iostream>
#include <string>

using namespace std;

// Node struct for Double Linked List
struct Node {
    string productName;
    int price;
    Node *prev;
    Node *next;
};

Node *head;
Node *tail;

// Initialize the double linked list
void init() {
    head = NULL;
    tail = NULL;
}
```

```
// Check if the list is empty
bool isEmpty() {
   return head == NULL;
}

// Insert at the end
void insertBelakang(string productName, int price) {
   Node *newNode = new Node;
   newNode->productName = productName;
   newNode->price = price;
   newNode->prev = NULL;
   newNode->next = NULL;
   if (isEmpty()) {
      head = tail = newNode;
   } else {
      tail->next = newNode;
      newNode->prev = tail;
      tail = newNode;
   }
}
```

```
// Insert at a specific position
void insertTengah(string productName, int price, int posisi) {
    if (posisi < 1) {
        cout << "Posisi di luar jangkauan" << endl;
        return;
    }

    Node *newNode = new Node;
    newNode->productName = productName;
    newNode->price = price;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (posisi == 1) {
        if (isEmpty()) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
        return;
    }
```

```
Node *current = head;
    int count = 1;
    while (current != NULL && count < posisi - 1) {
        current = current->next;
        count++;
    }

    if (current == NULL) {
        cout << "Posisi di luar jangkauan" << endl;
        delete newNode;
        return;
    }

    newNode->next = current->next;
    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    newNode->prev = current;
    current->next = newNode;

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

```
// Delete a node by product name
void hapusTengah(string productName) {
    if (isEmpty()) {
        cout << "List kosong!" << endl;
        return;
    }

    Node *current = head;
    while (current != NULL && current->productName != productName) {
        current = current->next;
    }

    if (current == NULL) {
        cout << "Nama produk tidak ditemukan!" << endl;
        return;
    }

    if (current->prev != NULL) {
        current->prev->next = current->next;
    } else {
        head = current->next;
    }

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

```
// Update product data by name
void ubahData(string oldProductName, string newProductName, int newPrice) {
    Node *current = head;
    while (current != NULL) {
        if (current->productName == oldProductName) {
            current->productName = newProductName;
            current->price = newPrice;
            return;
        }
        current = current->next;
    }
    cout << "Nama produk tidak ditemukan!" << endl;
}

// Display the list
void tampil() {
    Node *current = head;
    if (isEmpty()) {
        cout << "List masih kosong!" << endl;
        return;
    }
    while (current != NULL) {
        cout << current->productName << " " << current->price << endl;
        current = current->next;
    }
    cout << endl;
}
```

```
// Clear the entire list
void clearList() {
   Node *current = head;
   while (current != NULL) {
      Node *hapus = current;
      current = current->next;
      delete hapus;
   }
   head = tail = NULL;
   cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
   init();

   // Initial data
   insertBelakang("Originote", 60000);
   insertBelakang("Somethinc", 150000);
   insertBelakang("Skintific", 100000);
   insertBelakang("Wardah", 50000);
   insertBelakang("Hanasui", 30000);
   tampil();
```

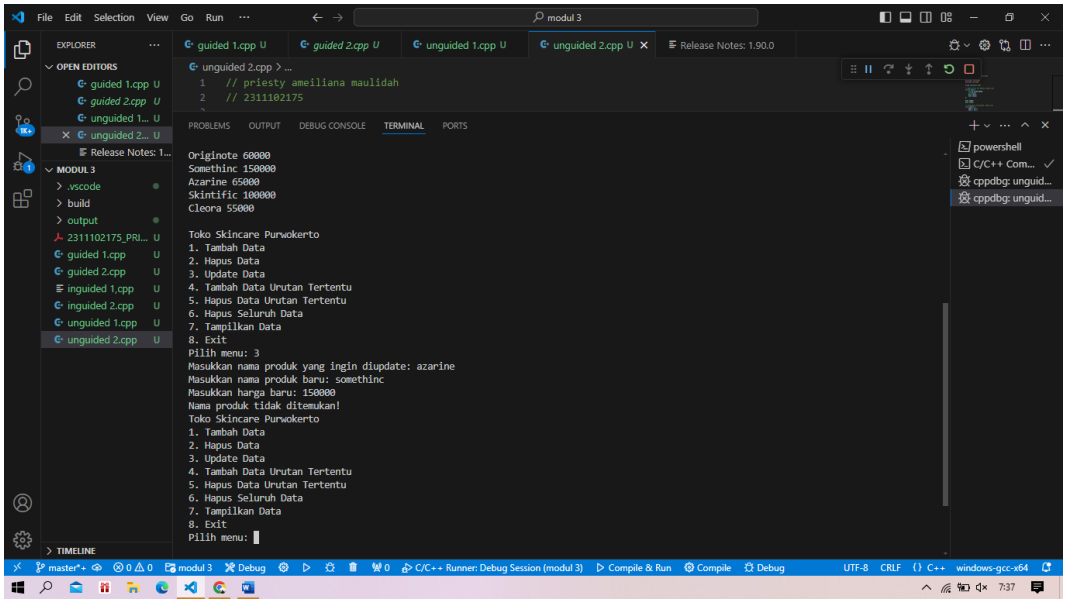
```
// Case 1: Insert Azarine between Somethinc and Skintific
    insertTengah("Azarine", 65000, 3);
    tampil();

    // Case 2: Delete product Wardah
    hapusTengah("Wardah");
    tampil();

    // Case 3: Update Hanasui to Cleora with price 55000
    ubahData("Hanasui", "Cleora", 55000);
    tampil();

    // Menu implementation
    int choice;
    string productName, oldProductName, newProductName;
    int price, posisi;
```

```
while (true) {
        cout << "Toko Skincare Purwokerto" << 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 menu: ";
        cin >> choice;

        switch (choice) {
           case 1:
              cout << "Masukkan nama produk: ";
              cin >> productName;
              cout << "Masukkan harga: ";
              cin >> price;
              insertBelakang(productName, price);
              break;
           case 2:
              cout << "Masukkan nama produk yang ingin dihapus: ";
              cin >> productName;
              hapusTengah(productName);
              break;
```

```
case 3:
            cout << "Masukkan nama produk yang ingin diupdate: ";
            cin >> oldProductName;
            cout << "Masukkan nama produk baru: ";
            cin >> newProductName;
            cout << "Masukkan harga baru: ";
            cin >> price;
            ubahData(oldProductName, newProductName, price);
            break;
        case 4:
            cout << "Masukkan nama produk: ";
            cin >> productName;
            cout << "Masukkan harga: ";
            cin >> price;
            cout << "Masukkan posisi: ";
            cin >> posisi;
            insertTengah(productName, price, posisi);
            break;
        case 5:
            cout << "Masukkan posisi produk yang ingin dihapus: ";
            cin >> posisi;
            // Finding the product at the given position and deleting it
            {
```

```
// Finding the product at the given position and deleting it
            {
               Node *current = head;
               int count = 1;
               while (current != NULL && count < posisi) {
                  current = current->next;
                  count++;
               }
               if (current != NULL) {
                  hapusTengah(current->productName);
               } else {
                  cout << "Posisi di luar jangkauan" << endl;
               }
            }
            break;
         case 6:
            clearList();
            break;
         case 7:
            tampil();
            break;
         case 8:
            return 0;
         default:
            cout << "Pilihan tidak valid!" << endl;
      }
   }
```

Screenshots output:



Deskripsi :

Struktur Data Linked List Berkait Ganda

Linked list berkait ganda adalah struktur data linier dinamis yang serupa dengan linked list biasa, tetapi setiap node memiliki dua pointer tambahan:

- **prev (sebelumnya): pointer yang menunjuk ke node sebelumnya di dalam linked list**
- **next (selanjutnya): pointer yang menunjuk ke node berikutnya di dalam linked list**

Fungsi-Fungsi Program

Program ini memiliki beberapa fungsi untuk menambah, menghapus, memperbarui, menampilkan, dan menghapus seluruh isi linked list:

- **init(): Meminisiasi linked list dengan membuat head dan tail bernilai NULL, menandakan linked list masih kosong.**
- **isEmpty(): Mengecek apakah linked list kosong. Mengembalikan nilai true jika kosong, sebaliknya false.**
- **insertBelakang(productName, price): Menambahkan node baru berisi nama produk (productName) dan harga (price) di akhir linked list.**
- **insertTengah(productName, price, posisi): Menambahkan node baru berisi nama produk (productName) dan harga (price) pada posisi tertentu di dalam linked list (posisi dimulai dari 1). Fungsi ini akan menampilkan pesan kesalahan jika posisi di luar jangkauan.**
- **hapusTengah(productName): Menghapus node yang memiliki nama produk (productName) yang diberikan dari linked list. Fungsi ini akan menampilkan pesan kesalahan jika linked list kosong atau nama produk tidak ditemukan.**
- **ubahData(oldProductName, newProductName, newPrice): Mengubah nama produk (oldProductName) dan harga (newPrice) dari node yang memiliki nama produk yang**

diberikan menjadi nama produk baru (newProductName). Fungsi ini akan menampilkan pesan kesalahan jika nama produk tidak ditemukan.

- **tampil(): Menampilkan seluruh isi linked list, yaitu nama produk dan harga dari setiap node.**
- **clearList(): Menghapus seluruh node di dalam linked list dan menyetel head dan tail menjadi NULL/**

E. Referensi

https://socs.binus.ac.id/2017/03/15/single-linked-list/