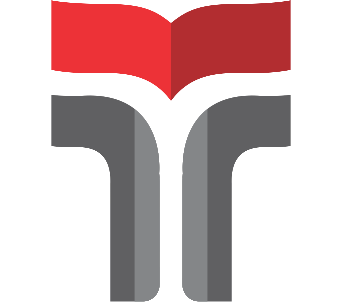
**LAPORAN PRAKTIKUM**

**MODUL 3**

**SINGLE AND DOUBLE LINKED LIST**

****

**Disusun oleh:**

**Tegar Bangkit Wijaya**

**NIM: 2311102027**

**Dosen Pengampu:**

Wahyu Andi Saputra S.PD., M Eng.

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**PURWOKERTO**

**2024**

**BAB I**

**TUJUAN PRAKTIKUM**

1. **TUJUAN PRAKTIKUM**
2. Mahasiswa memahami konsep Single dan Double Linked List
3. Mahasiswa mampu menerapkan Single dan Double Linked List ke dalam pemograman

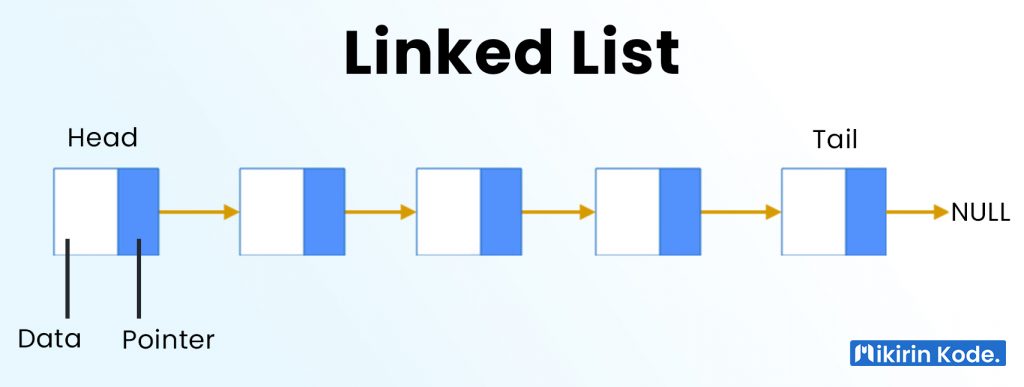
**BAB II**

**DASAR TEORI**

B.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.

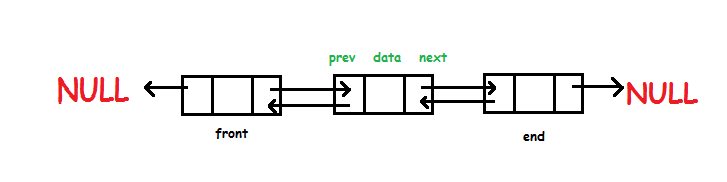


Dalam operasi Single Linked List, umumnya dilakukan operasi penambahan dan penghapusan simpul pada awal atau akhir daftar, serta pencarian dan pengambilan nilai pada simpul tertentu dalam daftar. Karena struktur data ini hanya memerlukan satu pointer untuk setiap simpul, maka Single Linked List. umumnya 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 Single Linked List. 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**

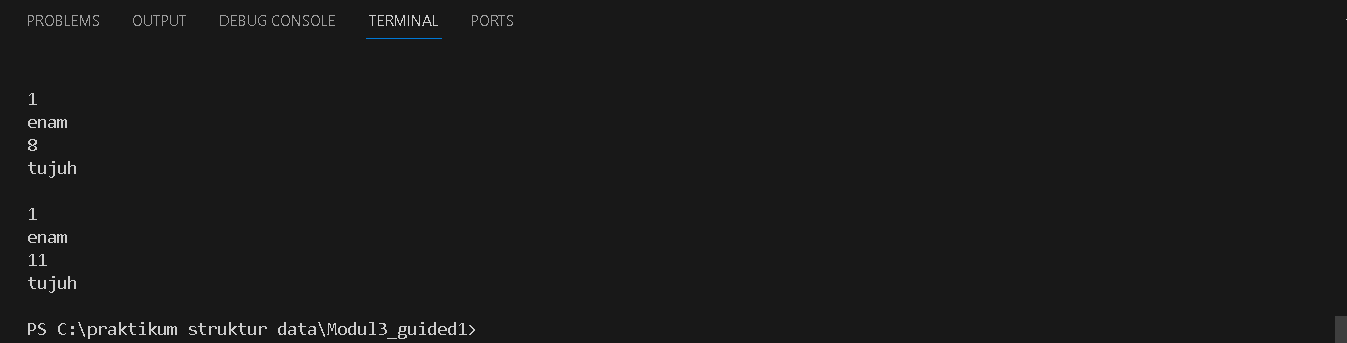
**GUIDED**

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 << endl;  cout << bantu->kata << endl;  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**

****

**Deskripsi program**

Program di atas adalah implementasi dari single linked list non-circular. Pada program ini, sebuah linked list dibuat dengan menggunakan struktur data `Node` yang memiliki dua komponen, yaitu `data` dan `kata`, serta sebuah pointer `next` yang menunjuk ke node selanjutnya. Program ini menyediakan fungsi-fungsi dasar untuk mengelola linked list seperti inisialisasi, pengecekan apakah linked list kosong, penambahan elemen di depan, di belakang, di tengah, penghapusan elemen di depan, di belakang, di tengah, serta pengubahan nilai elemen di depan, di belakang, dan di tengah. Fungsi `tampil()` digunakan untuk menampilkan isi linked list. Pada fungsi main()`, beberapa operasi seperti penambahan, penghapusan, dan pengubahan elemen dilakukan untuk menguji fungsi-fungsi yang telah dibuat.

1. **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)

{

{

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;

}

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;

}

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)

{

case 1:

{

{

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;

}

}

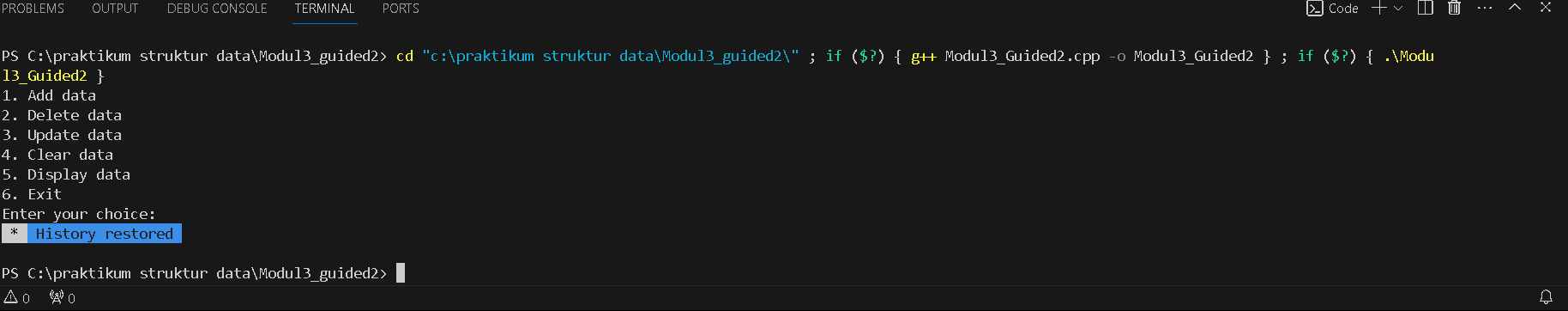
}

}

return 0;

}

**Screenshot program**

****

**Deskripsi program**

Program di atas merupakan implementasi dari struktur data doubly linked list. Doubly linked list adalah tipe struktur data yang setiap nodenya memiliki dua pointer, yaitu satu yang menunjuk ke node sebelumnya (prev) dan satu yang menunjuk ke node selanjutnya (next). Kelas `Node` digunakan untuk merepresentasikan sebuah node dalam doubly linked list. Setiap node memiliki dua data, yaitu data yang bertipe integer (`int`) dan kata yang bertipe string (`string`), serta dua pointer yang menunjuk ke node sebelumnya (`prev`) dan node selanjutnya (`next`). Kelas `DoublyLinkedList` digunakan untuk mengimplementasikan operasi-operasi dasar pada doubly linked list, seperti menambahkan node di awal (`push`), menghapus node dari awal (`pop`), mengubah nilai node berdasarkan nilai lama (`update`), menghapus semua node dari list (`deleteAll`), dan menampilkan semua data dalam list (`display`). Fungsi `main()` adalah fungsi utama yang memberikan menu kepada pengguna untuk melakukan operasi-operasi tersebut. Setiap pilihan yang dibuat oleh pengguna akan memanggil fungsi yang sesuai di dalam kelas `DoublyLinkedList`. Program akan terus berjalan hingga pengguna memilih untuk keluar (`Exit`).

**LATIHAN KELAS - UNGUIDED**

1. **Unguided 1**

**Source code**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  // Deklarasi Struct Node  struct Node  {  string nama;  int usia;  Node \*next;  };  Node \*head = nullptr;  // Fungsi untuk menambahkan data di depan linked list  void insertDepan(string nama, int usia)  {  Node \*newNode = new Node;  newNode->nama = nama;  newNode->usia = usia;  newNode->next = head;  head = newNode;  }  // Fungsi untuk menambahkan data di belakang linked list  void insertBelakang(string nama, int usia)  {  Node \*newNode = new Node;  newNode->nama = nama;  newNode->usia = usia;  newNode->next = nullptr;  if (head == nullptr)  {  head = newNode;  return;  }  Node \*temp = head;  while (temp->next != nullptr)  {  temp = temp->next;  }  temp->next = newNode;  }  // Fungsi untuk menambahkan data di tengah linked list  void insertTengah(string nama, int usia, string nama\_sebelumnya)  {  Node \*newNode = new Node;  newNode->nama = nama;  newNode->usia = usia;  Node \*temp = head;  while (temp != nullptr)  {  if (temp->nama == nama\_sebelumnya)  {  newNode->next = temp->next;  temp->next = newNode;  return;  }  temp = temp->next;  }  }  // Fungsi untuk menampilkan seluruh data  void tampilkanData()  {  Node \*temp = head;  while (temp != nullptr)  {  cout << temp->nama << " " << temp->usia << endl;  temp = temp->next;  }  }  int main()  {  string namaAnda;  int usiaAnda;  cout << "Masukkan nama anda: ";  getline(cin, namaAnda);  cout << "Masukkan usia anda: ";  cin >> usiaAnda;  cin.ignore();  insertDepan("Tegar", 19);  insertBelakang("John", 19);  insertBelakang("Jane", 20);  insertBelakang("Michael", 18);  insertBelakang("Yusuke", 19);  insertBelakang("Akechi", 20);  insertBelakang("Hoshino", 18);  insertBelakang("Karin", 18);  cout << "Data Mahasiswa:" << endl;  tampilkanData();  return 0;  } |

**Screenshoot program**

****

**Deskripsi program**

Program di atas merupakan implementasi dari linked list sederhana menggunakan bahasa pemrograman C++. Program ini memiliki beberapa fungsi untuk menambahkan data ke linked list, yaitu `insertDepan()` untuk menambahkan data di depan linked list, `insertBelakang()` untuk menambahkan data di belakang linked list, dan `insertTengah()` untuk menambahkan data di tengah linked list setelah node dengan nama tertentu.

Dalam program ini, setiap node dalam linked list direpresentasikan oleh struktur `Node`, yang memiliki dua atribut yaitu `nama` yang bertipe `string` untuk menyimpan nama mahasiswa dan `usia` yang bertipe `int` untuk menyimpan usia mahasiswa. Setiap node juga memiliki pointer `next` yang menunjuk ke node selanjutnya dalam linked list.

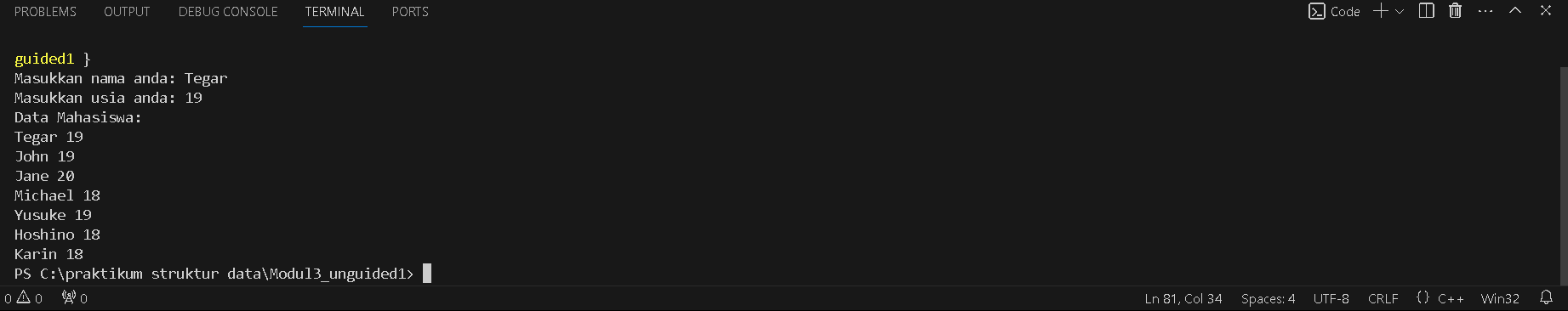
Fungsi `insertDepan()` digunakan untuk menambahkan data di depan linked list. Fungsi ini membuat sebuah node baru dengan data yang diberikan, lalu mengatur pointer `next` dari node baru tersebut untuk menunjuk ke head sebelumnya. Kemudian, head diupdate menjadi node baru.

Fungsi `insertBelakang()` digunakan untuk menambahkan data di belakang linked list. Fungsi ini membuat sebuah node baru dengan data yang diberikan, lalu mencari node terakhir dalam linked list dengan melakukan iterasi sampai menemukan node yang `next`-nya `nullptr`. Setelah itu, node terakhir tersebut diupdate untuk menunjuk ke node baru.

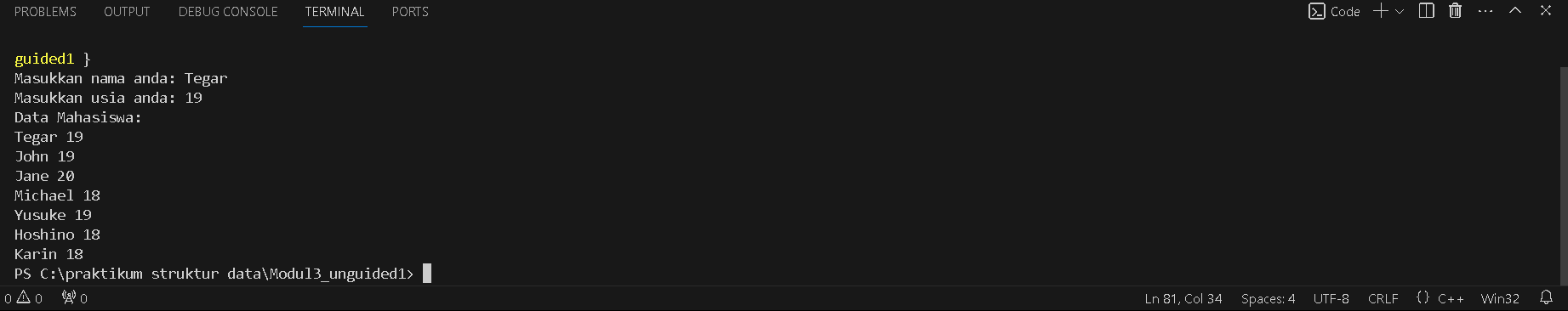
Fungsi `insertTengah()` digunakan untuk menambahkan data di tengah linked list setelah node dengan nama tertentu. Fungsi ini mencari node dengan nama yang sesuai, lalu menambahkan node baru di antara node tersebut dan node selanjutnya.

Fungsi `tampilkanData()` digunakan untuk menampilkan seluruh data yang tersimpan dalam linked list. Program utama `main()` meminta pengguna untuk memasukkan nama dan usia, lalu menambahkan beberapa data mahasiswa ke linked list menggunakan fungsi-fungsi yang telah dibuat, dan akhirnya menampilkan seluruh data yang tersimpan dalam linked list.

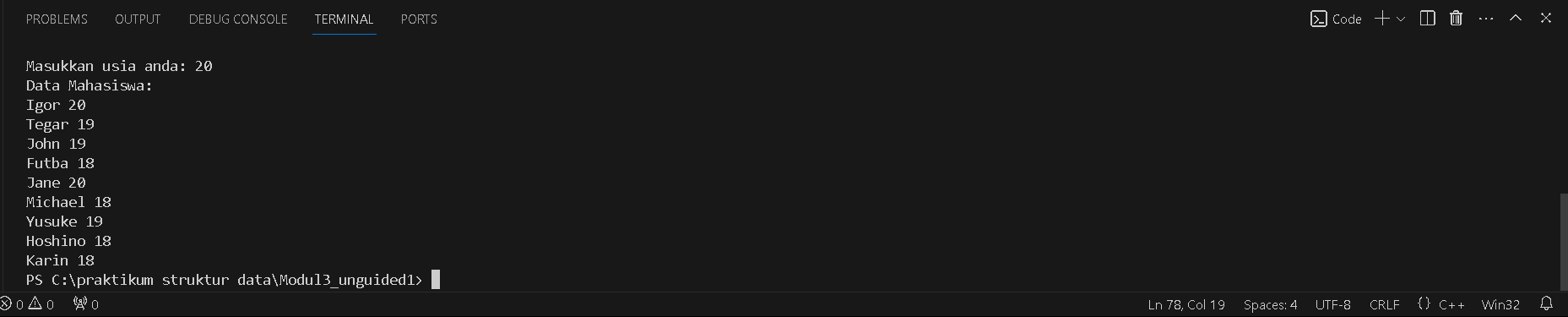
1. **Screenshoot Hapus Data Akechi**

****

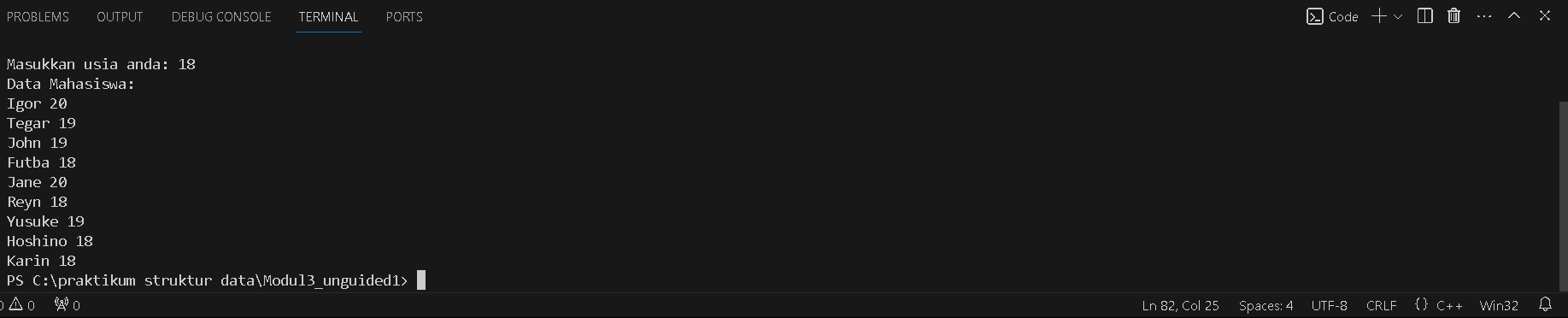
1. **Screenshoot Tambah Data Futba 18 diantara John dan Jane**

****

1. **Screenshoot Tambah Data Igor 20 diawal**

****

1. **Screenshoot Ubah data Michael menjadi Reyn 18**

****

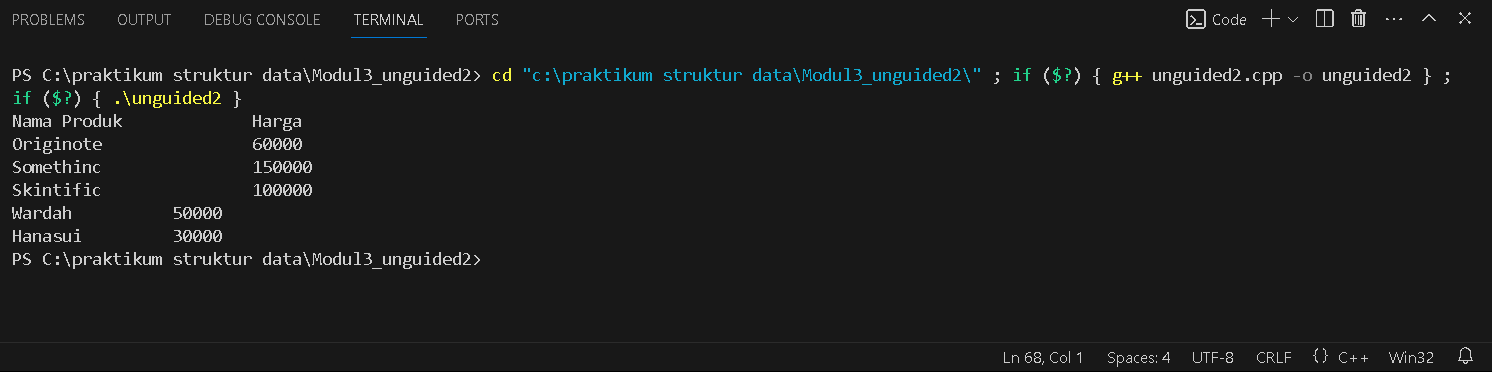
1. **Screenshoot Tampilkan seluruh data**

****

1. **Unguided 2**

|  |
| --- |
| // Unguided 2  #include <iostream>  #include <string>  using namespace std;  struct Node  {  string namaProduk;  int harga;  Node \*prev;  Node \*next;  };  class DoublyLinkedList  {  private:  Node \*head;  Node \*tail;  public:  DoublyLinkedList() : head(nullptr), tail(nullptr) {}  void insertBelakang(string namaProduk, int harga)  {  Node \*newNode = new Node;  newNode->namaProduk = namaProduk;  newNode->harga = harga;  newNode->prev = nullptr;  newNode->next = nullptr;  if (head == nullptr)  {  head = tail = newNode;  }  else  {  tail->next = newNode;  newNode->prev = tail;  tail = newNode;  }  }  void display()  {  Node \*current = head;  while (current != nullptr)  {  cout << current->namaProduk << "\t" << current->harga << endl;  current = current->next;  }  }  };  int main()  {  DoublyLinkedList list;  // Menambahkan data  list.insertBelakang("Originote", 60000);  list.insertBelakang("Somethinc", 150000);  list.insertBelakang("Skintific", 100000);  list.insertBelakang("Wardah", 50000);  list.insertBelakang("Hanasui", 30000);  cout << "Nama Produk\tHarga" << endl;  list.display();  return 0;  } |

**Screenshoot program**

****

**Deskripsikan program**

Program ini merupakan implementasi dari doubly linked list yang digunakan untuk menyimpan data produk. Struktur data `Node` digunakan untuk merepresentasikan setiap elemen dalam linked list, yang terdiri dari nama produk (`namaProduk`) dengan tipe data `string`, harga produk (`harga`) dengan tipe data `int`, serta dua pointer yaitu `prev` yang menunjukkan ke node sebelumnya dan `next` yang menunjukkan ke node selanjutnya.

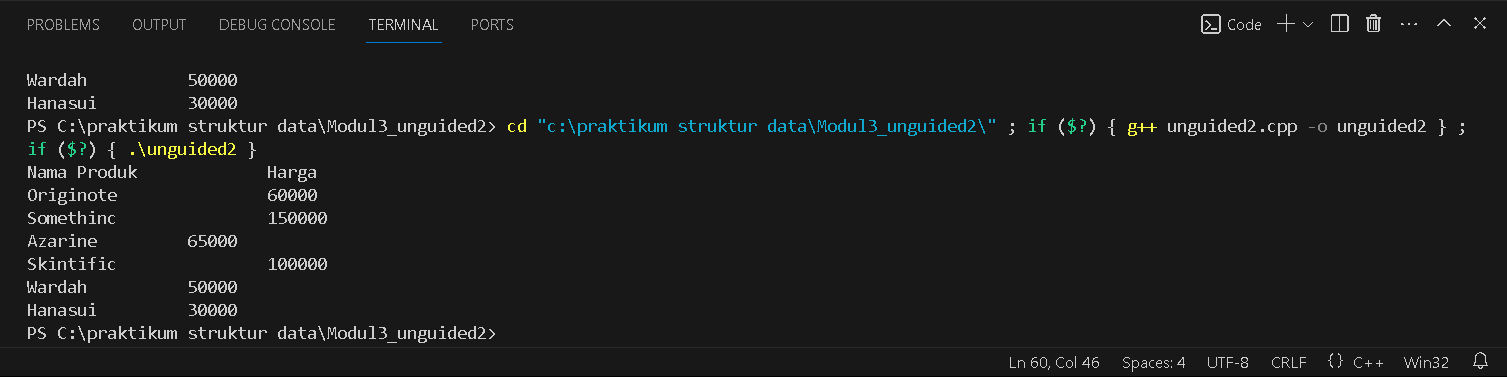
Kelas `DoublyLinkedList` memiliki dua pointer yaitu `head` dan `tail` yang menunjukkan node pertama dan terakhir dalam linked list. Konstruktor kelas ini menginisialisasi kedua pointer tersebut menjadi `nullptr`.

Metode `insertBelakang` digunakan untuk menambahkan sebuah node baru di belakang linked list. Metode ini menerima dua parameter, yaitu nama produk (`namaProduk`) dan harga produk (`harga`). Sebuah node baru akan dibuat dengan nilai `namaProduk` dan `harga` sesuai dengan parameter, serta pointer `prev` dan `next` diatur menjadi `nullptr`. Jika linked list masih kosong, maka `head` dan `tail` akan menunjuk ke node baru tersebut. Jika tidak, node baru akan ditambahkan setelah node terakhir dan `tail` akan diperbarui.

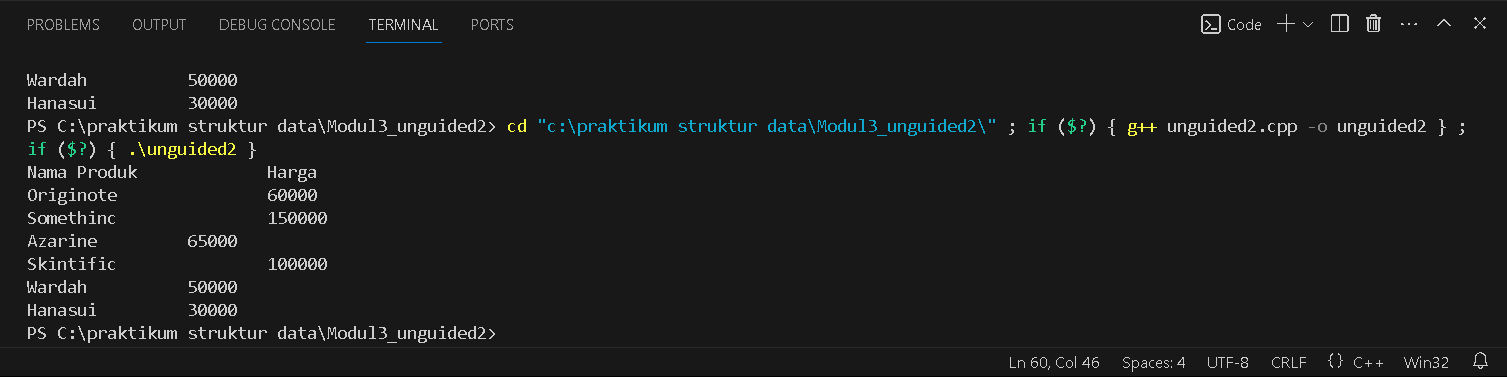
Metode `display` digunakan untuk menampilkan seluruh data produk yang tersimpan dalam linked list. Metode ini akan menelusuri linked list dari awal (`head`) hingga akhir (`tail`), kemudian menampilkan nama produk dan harganya secara berurutan.

Di dalam fungsi `main`, sebuah objek dari kelas `DoublyLinkedList` dibuat dan beberapa data produk ditambahkan menggunakan metode `insertBelakang`. Setelah itu, seluruh data produk ditampilkan menggunakan metode `display`.

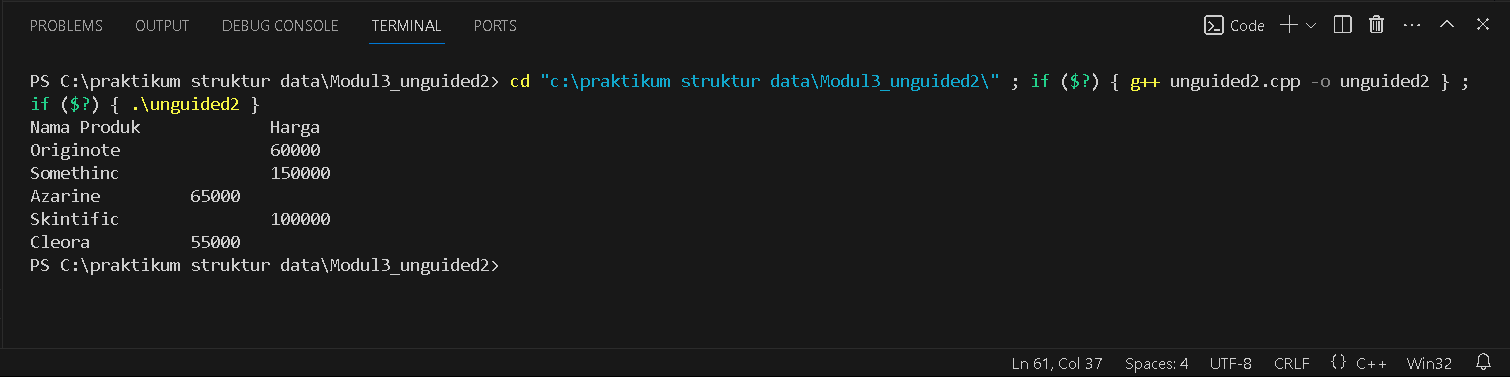
**B. Screenshoot produk Azarin dengan harga 6500 diantara Somethinc dan Skintific**

****

**C. Screenshot Hapus produk Wardah**

****

**D. Screenshoot update produk Hanasui menjadi Cleora dengan harga 55000**

****

**E. Screenshoot Tampilkan menu seperti ini**

1. Tambah Data

2. Hapus Data

3. Update Data

4. Tambah Data Urutan Tertentu

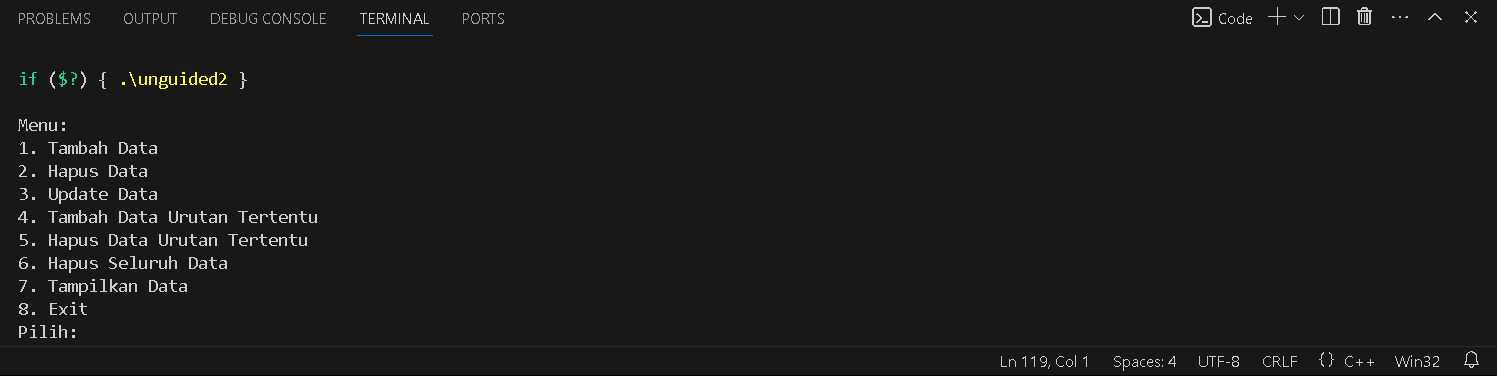
5. Hapus Data Urutan Tertentu

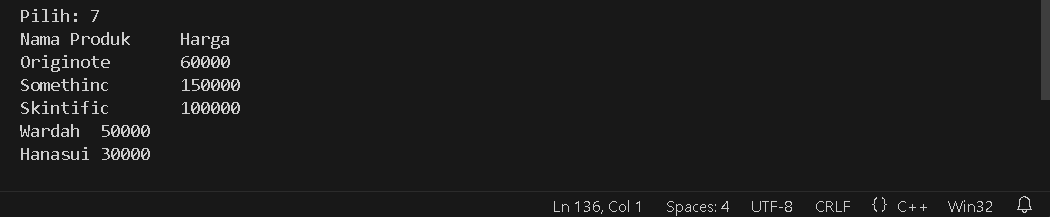
6. Hapus Seluruh Data

7. Tampilkan Data

8. Exit

**Screenshot**

****

****

**BAB IV**

**KESIMPULAN**

Program yang telah dibuat adalah sebuah implementasi dari struktur data yang kuat dan fleksibel untuk menyimpan dan mengelola data dalam urutan tertentu. Berbeda dengan single linked list, double linked list memiliki keunggulan karena setiap node memiliki dua pointer, yaitu pointer ke node sebelumnya (prev) dan pointer ke node selanjutnya (next). Hal ini memungkinkan untuk melakukan traversing ke depan dan ke belakang dengan mudah. Program ini memberikan kemampuan kepada pengguna untuk menambah, menghapus, mengupdate, dan menampilkan data dengan berbagai fungsionalitas yang disediakan. Dengan adanya program ini, pengguna dapat mengelola data toko skincare dengan lebih efisien dan efektif, sehingga memudahkan dalam operasional sehari-hari. Kesimpulannya, program Double Linked List ini memberikan solusi yang efektif dalam manajemen data dengan struktur yang terorganisir dan mudah diakses.