**LAPORAN PRAKTIKUM**

**Modul III**

**Single dan Double Linked List**

Logo

Description automatically generated

**Disusun oleh:**

Muhammad Irsyad : **2211102048**

**Dosen**

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

**PROGRAM STUDI S1 INFORMATIKA**

**FAKULTAS INFORMATIKA**

**INSTITUT TEKNOLOGI TELKOM PURWOKERTO**

**2023**

# BAB I

## Tujuan Pembelajaran

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

# BAB II

# Dasar Teori

1. Single Linked List Linked List merupakan suatu bentuk struktur data yang berisi kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung, dinamis, dan terbatas. Untuk menghubungkan satu node dengan node yang lainnya Linked List menggunakan pointer sebagai penunjuk node selanjutnya. Node sendiri merupakan sebuah struct yang terdiri dari beberapa field, minimal ada 2 buah field yaitu field untuk isi dari struct datanya sendiri, dan 1 field arbitrary bertipe pointer sebagai penunjuk node selanjutnya. Linked List merupakan suatu bentuk struktur data yang berisi kumpulan data yang disebut sebagai node yang tersusun secara sekuensial, saling sambung menyambung, dinamis, dan terbatas. Untuk menghubungkan satu node dengan node yang lainnya Linked List menggunakan pointer sebagai penunjuk node selanjutnya. Node sendiri merupakan sebuah struct yang terdiri dari beberapa field, minimal ada 2 buah field yaitu field untuk isi dari struct datanya sendiri, dan 1 field arbitrary bertipe pointer sebagai penunjuk node selanjutnya.

Diagram

Description automatically generated

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.

Chart, box and whisker chart

Description automatically generated

1. 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 memiliki pointer penunjuk 2 arah, yakni ke arah node sebelum (previous/prev) dan node sesudah (next). Representasi sebuah doubly linked list dapat dilihat pada gambar berikut ini:

Diagram

Description automatically generated

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

## LATIHAN KELAS – GUIDED

### Guided 1

**Source Code**

|  |
| --- |
| #include <iostream>  using namespace std;  /// PROGRAM SINGLE LINKED LIST NON-CIRCULAR  // Deklarasi Struct Node  struct Node  {  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;  } |

**Screenshot Program**

A screenshot of a computer

Description automatically generated with medium confidence

**Deskripsi Program**

Kode program di atas merupakan implementasi dari linked list berbasis single linked list non-circular dalam bahasa C++. Berikut adalah deskripsi fungsi-fungsi yang ada dalam kode program tersebut:

1. **init()**: Fungsi ini digunakan untuk menginisialisasi linked list dengan mengatur **head** dan **tail** menjadi **NULL**.
2. **isEmpty()**: Fungsi ini digunakan untuk memeriksa apakah linked list kosong atau tidak. Jika **head** bernilai **NULL**, maka dikembalikan **true**, jika tidak maka dikembalikan **false**.
3. **insertDepan(int nilai)**: Fungsi ini digunakan untuk menambahkan sebuah node baru dengan nilai **nilai** di bagian depan (head) linked list. Node baru akan menjadi node pertama (head) jika linked list kosong, atau akan menjadi node sebelum node yang sudah ada jika linked list tidak kosong.
4. **insertBelakang(int nilai)**: Fungsi ini digunakan untuk menambahkan sebuah node baru dengan nilai **nilai** di bagian belakang (tail) linked list. Node baru akan menjadi node terakhir (tail) jika linked list kosong, atau akan menjadi node setelah node yang sudah ada jika linked list tidak kosong.
5. **hitungList()**: Fungsi ini digunakan untuk menghitung jumlah node yang ada dalam linked list dan mengembalikan nilai jumlah tersebut.
6. **insertTengah(int data, int posisi)**: Fungsi ini digunakan untuk menambahkan sebuah node baru dengan nilai **data** pada posisi **posisi** dalam linked list. Posisi dihitung dari 1 sebagai posisi pertama (head) hingga **hitungList()** sebagai posisi terakhir (tail).
7. **hapusDepan()**: Fungsi ini digunakan untuk menghapus node pertama (head) dalam linked list.
8. **hapusBelakang()**: Fungsi ini digunakan untuk menghapus node terakhir (tail) dalam linked list.
9. **hapusTengah(int posisi)**: Fungsi ini digunakan untuk menghapus node pada posisi **posisi** dalam linked list. Posisi dihitung dari 1 sebagai posisi pertama (head) hingga **hitungList()** sebagai posisi terakhir (tail).
10. **ubahDepan(int data)**: Fungsi ini digunakan untuk mengubah nilai data pada node pertama (head) dalam linked list.
11. **ubahTengah(int data, int posisi)**: Fungsi ini digunakan untuk mengubah nilai data pada node pada posisi **posisi** dalam linked list. Posisi dihitung dari 1 sebagai posisi pertama (head) hingga **hitungList()** sebagai posisi terakhir (tail).

### Guided 2

**Source Code**

|  |
| --- |
| #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 Program**

Text

Description automatically generated

**Deskripsi Program**

Kode program di atas merupakan implementasi dari sebuah Doubly Linked List (Daftar Berantai Ganda) dalam bahasa C++. Berikut adalah deskripsi lebih rinci dari kode program tersebut:

1. Deklarasi dan definisi kelas Node:
   * Kelas Node memiliki tiga anggota data: int data untuk menyimpan data dalam node, Node \*prev untuk menyimpan alamat node sebelumnya dalam daftar, dan Node \*next untuk menyimpan alamat node berikutnya dalam daftar.
2. Deklarasi dan definisi kelas DoublyLinkedList:
   * Kelas DoublyLinkedList memiliki dua anggota data: Node \*head untuk menyimpan alamat node pertama dalam daftar, dan Node \*tail untuk menyimpan alamat node terakhir dalam daftar. Constructor kelas DoublyLinkedList digunakan untuk menginisialisasi head dan tail menjadi nullptr, menandakan daftar masih kosong.
   * Metode push digunakan untuk menambahkan node baru dengan data tertentu ke awal daftar. Node baru dibuat dan diatur prev-nya menjadi nullptr, dan next-nya menjadi node yang saat ini menjadi head. Jika daftar tidak kosong sebelumnya, prev dari head saat ini diatur menjadi node baru, jika tidak, tail diatur menjadi node baru.
   * Metode pop digunakan untuk menghapus node pertama dalam daftar. Head saat ini diatur menjadi next node dari head saat ini, dan prev dari head saat ini diatur menjadi nullptr. Node yang dihapus kemudian didealokasi dari memori.
   * Metode update digunakan untuk mengganti data lama dalam daftar dengan data baru. Metode ini melakukan pencarian data lama dalam daftar, dan jika ditemukan, data dalam node yang sesuai diubah menjadi data baru.
   * Metode deleteAll digunakan untuk menghapus semua node dalam daftar dan mengosongkan daftar. Setiap node dihapus dan didealokasi dari memori, dan head dan tail diatur menjadi nullptr.
   * Metode display digunakan untuk menampilkan data dalam daftar secara berurutan dari head ke tail.
3. Fungsi main:
   * Fungsi main digunakan untuk mengatur menu dan interaksi pengguna dengan daftar. Pengguna dapat memilih untuk menambahkan data, menghapus data, memperbarui data, menghapus semua data, menampilkan data, atau keluar dari program. Pilihan pengguna kemudian diteruskan ke metode yang sesuai dalam kelas DoublyLinkedList untuk menjalankan operasi yang diminta. Loop while(true) digunakan untuk menjalankan menu dalam loop tak terbatas, sampai pengguna memilih untuk keluar dari program dengan memilih opsi 6.

## TUGAS – UNGUIDED

### Unguided 1

**Source Code**

|  |
| --- |
| #include <iostream>  using namespace std;  struct Mahasiswa  {  string nama;  int usia;  Mahasiswa \*next;  };  class LinkedList  {  private:  Mahasiswa \*head;  public:  LinkedList()  {  head = NULL;  }  void tambahMahasiswaAwal()  {  Mahasiswa \*new\_mahasiswa = new Mahasiswa;  cout << "Masukkan nama mahasiswa: ";  cin >> new\_mahasiswa->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> new\_mahasiswa->usia;  new\_mahasiswa->next = NULL;  if (head == NULL)  {  head = new\_mahasiswa;  }  else  {  Mahasiswa \*current = head;  while (current->next != NULL)  {  current = current->next;  }  current->next = new\_mahasiswa;  }  cout << "Mahasiswa berhasil ditambahkan." << endl;  }  void tambahMahasiswa()  {  int pilihan;  cout << "1. Insert depan\n";  cout << "2. Insert belakang\n";  cout << "3. Insert tengah\n";  cout << "Masukkan pilihan: ";  cin >> pilihan;  if (pilihan == 1)  {  Mahasiswa \*new\_mahasiswa = new Mahasiswa;  cout << "Masukkan nama mahasiswa: ";  cin >> new\_mahasiswa->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> new\_mahasiswa->usia;  new\_mahasiswa->next = head;  head = new\_mahasiswa;  }  else if (pilihan == 2)  {  Mahasiswa \*new\_mahasiswa = new Mahasiswa;  cout << "Masukkan nama mahasiswa: ";  cin >> new\_mahasiswa->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> new\_mahasiswa->usia;  new\_mahasiswa->next = NULL;  Mahasiswa \*current = head;  while (current->next != NULL)  {  current = current->next;  }  current->next = new\_mahasiswa;  }  else if (pilihan == 3)  {  int posisi;  cout << "Masukkan posisi: ";  cin >> posisi;  if (posisi < 1)  {  cout << "Posisi harus lebih besar dari 0." << endl;  }  else if (posisi == 1)  {  Mahasiswa \*new\_mahasiswa = new Mahasiswa;  cout << "Masukkan nama mahasiswa: ";  cin >> new\_mahasiswa->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> new\_mahasiswa->usia;  new\_mahasiswa->next = head;  head = new\_mahasiswa;  }  else  {  Mahasiswa \*new\_mahasiswa = new Mahasiswa;  cout << "Masukkan nama mahasiswa: ";  cin >> new\_mahasiswa->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> new\_mahasiswa->usia;  Mahasiswa \*current = head;  for (int i = 1; i < posisi - 1; i++)  {  if (current->next == NULL)  {  cout << "Posisi tidak valid." << endl;  return;  }  current = current->next;  }  new\_mahasiswa->next = current->next;  current->next = new\_mahasiswa;  }  }  else  {  cout << "Pilihan tidak valid." << endl;  }  }  void hapusMahasiswa()  {  if (head == NULL)  {  cout << "Linked list kosong." << endl;  }  else  {  int posisi;  cout << "Masukkan posisi: ";  cin >> posisi;  if (posisi < 1)  {  cout << "Posisi harus lebih besar dari 0." << endl;  }  else if (posisi == 1)  {  Mahasiswa \*temp = head;  head = head->next;  delete temp;  }  else  {  Mahasiswa \*current = head;  for (int i = 1; i < posisi - 1; i++)  {  if (current->next == NULL)  {  cout << "Posisi tidak valid." << endl;  return;  }  current = current->next;  }  Mahasiswa \*temp = current->next;  current->next = temp->next;  delete temp;  }  cout << "Mahasiswa berhasil dihapus." << endl;  }  }  void tampilMahasiswa()  {  if (head == NULL)  {  cout << "Linked list kosong." << endl;  }  else  {  Mahasiswa \*current = head;  while (current != NULL)  {  cout << "Nama: " << current->nama << ", Usia: " << current->usia << endl;  current = current->next;  }  }  }  void ubahMahasiswa2()  {  if (head == NULL)  {  cout << "Linked list kosong." << endl;  }  else  {  int posisi;  cout << "Masukkan posisi: ";  cin >> posisi;  if (posisi < 1)  {  cout << "Posisi harus lebih besar dari 0." << endl;  }  else if (posisi == 1)  {  // ubah depan  cout << "Masukkan nama mahasiswa: ";  cin >> head->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> head->usia;  }  else  {  Mahasiswa \*current = head;  for (int i = 1; i < posisi; i++)  {  if (current == NULL)  {  cout << "Posisi tidak valid." << endl;  return;  }  current = current->next;  }  if (current == NULL)  {  cout << "Posisi tidak valid." << endl;  }  else  {  cout << "Masukkan nama mahasiswa: ";  cin >> current->nama;  cout << "Masukkan usia mahasiswa: ";  cin >> current->usia;  }  }  cout << "Mahasiswa berhasil diubah." << endl;  }  }  };  int main()  {  LinkedList linkedList;  int pilihan;  linkedList.tambahMahasiswaAwal();  do  {  cout << "1. Tambah Data Mahasiswa Diposisi Tertentu\n";  cout << "2. Hapus Data Mahasiswa\n";  cout << "3. Tampilkan Data\n";  cout << "4. Ganti Data Mahasiswa\n";  cout << "5. Exit\n";  cout << "Masukkan pilihan: ";  cin >> pilihan;  switch (pilihan)  {  case 1:  linkedList.tambahMahasiswa();  break;  case 2:  linkedList.hapusMahasiswa();  break;  case 3:  linkedList.tampilMahasiswa();  break;  case 4:  linkedList.ubahMahasiswa2();  break;  case 5:  cout << "Terima Kasih" << endl;  break;  default:  cout << "Pilihan tidak valid." << endl;  break;  }  } while (pilihan != 5);  return 0;  } |

**Screenshot Program**

Graphical user interface, text, application, Teams

Description automatically generated

**Deskripsi Program**

1. Masukkan data sesuai urutan berikut. (Gunakan insert depan, belakang atau tengah). Data pertama yang dimasukkan adalah nama dan usia anda.

[Nama\_anda] [Usia\_anda]

Budi 19

Carol 20

Ann 18

Yusuke 19

Akechi 20

Hoshino 18

Karin 18

Graphical user interface, text, application, Teams

Description automatically generated

Graphical user interface, text, application, Teams

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Hapus data Akechi

Graphical user interface, text, application, Teams

Description automatically generated

1. Tambahkan data berikut diantara Carol dan Ann : Futaba 18

Graphical user interface, text, application, Teams

Description automatically generated

1. Tambahlan data berikut diawal : Igor 20

Graphical user interface, text, application, Teams

Description automatically generated

1. Ubah data Carol menjadi : Reyn 18

Graphical user interface, text, application, Teams

Description automatically generated

1. Tampilkan seluruh dataGraphical user interface, text, application, Teams

   Description automatically generated

### Unguided 2

**Source Code**

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  class Skincare  {  public:  string namaProduk;  int harga;  Skincare \*prev;  Skincare \*next;  // Constructor  Skincare(string \_namaProduk, int \_harga)  {  namaProduk = \_namaProduk;  harga = \_harga;  prev = nullptr;  next = nullptr;  }  };  class TokoSkincare  {  private:  Skincare \*head;  public:  // Constructor  TokoSkincare()  {  head = nullptr;  }  // Destructor  ~TokoSkincare()  {  // Hapus semua node pada double linked list saat objek dihapus  Skincare \*current = head;  while (current != nullptr)  {  Skincare \*next = current->next;  delete current;  current = next;  }  }  // Menambahkan data pada akhir double linked list  void tambahData(string namaProduk, int harga)  {  Skincare \*newSkincare = new Skincare(namaProduk, harga);  if (head == nullptr)  {  head = newSkincare;  }  else  {  Skincare \*current = head;  while (current->next != nullptr)  {  current = current->next;  }  current->next = newSkincare;  newSkincare->prev = current;  }  cout << "Data berhasil ditambahkan." << endl;  }  // Menghapus data dari double linked list  void hapusData(string namaProduk)  {  if (head == nullptr)  {  cout << "Data kosong." << endl;  return;  }  Skincare \*current = head;  while (current != nullptr && current->namaProduk != namaProduk)  {  current = current->next;  }  if (current == nullptr)  {  cout << "Data tidak ditemukan." << endl;  }  else  {  if (current->prev != nullptr)  {  current->prev->next = current->next;  }  else  {  head = current->next;  }  if (current->next != nullptr)  {  current->next->prev = current->prev;  }  delete current;  cout << "Data berhasil dihapus." << endl;  }  }  // Mengupdate data dalam double linked list  void updateData(string namaLama)  {  if (head == NULL)  {  cout << "Linked list kosong!" << endl;  }  else  {  Skincare \*current = head;  while (current != NULL && current->namaProduk != namaLama)  {  current = current->next;  }  if (current == NULL)  {  cout << "Data tidak ditemukan!" << endl;  }  else  {  string namaBaru;  int harga;  cout << "Masukkan nama baru: ";  cin >> namaBaru;  current->namaProduk = namaBaru;  cout << "Masukkan harga baru: ";  cin >> harga;  current->harga = harga;  cout << "Data berhasil diubah!" << endl;  }  }  /\* if (head == nullptr)  {  cout << "Data kosong." << endl;  return;  }  Skincare \*current = head;  while (current != nullptr && current->namaProduk != namaProduk)  {  current = current->next;  }  if (current == nullptr)  {  cout << "Data tidak ditemukan." << endl;  }  else  {  current->harga = harga;  cout << "Data berhasil diupdate." << endl;  } \*/  }  // Menambahkan data pada urutan tertentu dalam double linked list  void tambahDataUrutanTertentu(string namaProduk, int harga, int urutan)  {  Skincare \*newSkincare = new Skincare(namaProduk, harga);  if (head == nullptr)  {  head = newSkincare;  }  else  {  Skincare \*current = head;  int count = 1;  while (current->next != nullptr && count < urutan - 1)  {  current = current->next;  count++;  }  if (count < urutan - 1)  {  cout << "Urutan tertentu tidak valid." << endl;  return;  }  newSkincare->prev = current;  newSkincare->next = current->next;  if (current->next != nullptr)  {  current->next->prev = newSkincare;  }  current->next = newSkincare;  }  cout << "Data berhasil ditambahkan pada urutan tertentu." << endl;  }  // Menghapus data pada urutan tertentu dalam double linked list  void hapusDataUrutanTertentu(int urutan)  {  if (head == nullptr)  {  cout << "Data kosong." << endl;  return;  }  Skincare \*current = head;  int count = 1;  while (current->next != nullptr && count < urutan)  {  current = current->next;  count++;  }  if (count < urutan)  {  cout << "Urutan tertentu tidak valid." << endl;  }  else  {  if (current->prev != nullptr)  {  current->prev->next = current->next;  }  else  {  head = current->next;  }  if (current->next != nullptr)  {  current->next->prev = current->prev;  }  delete current;  cout << "Data pada urutan tertentu berhasil dihapus." << endl;  }  }  // Menghapus seluruh data dalam double linked list  void hapusSeluruhData()  {  Skincare \*current = head;  while (current != nullptr)  {  Skincare \*next = current->next;  delete current;  current = next;  }  head = nullptr;  cout << "Seluruh data berhasil dihapus." << endl;  }  // Menampilkan data dalam double linked list  void tampilkanData()  {  if (head == nullptr)  {  cout << "Data kosong." << endl;  return;  }  cout << "Data di dalam toko skincare: " << endl;  Skincare \*current = head;  while (current != nullptr)  {  cout << "Nama Produk: " << current->namaProduk << ", Harga: " << current->harga << endl;  current = current->next;  }  }  };  int main()  {  TokoSkincare tokoSkincare;  int pilihan;  string namaProduk;  int harga;  int urutan;  do  {  cout << "======================================" << endl;  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 << "======================================" << endl;  cout << "Pilih menu: ";  cin >> pilihan;  string namaLama;  switch (pilihan)  {  case 1:  cout << "Masukkan Nama Produk: ";  cin.ignore();  getline(cin, namaProduk);  cout << "Masukkan Harga: ";  cin >> harga;  tokoSkincare.tambahData(namaProduk, harga);  break;  case 2:  cout << "Masukkan Nama Produk yang ingin dihapus: ";  cin.ignore();  getline(cin, namaProduk);  tokoSkincare.hapusData(namaProduk);  break;  case 3:  cout << "Masukkan nama yang ingin diubah: ";  cin.ignore();  getline(cin, namaLama);  tokoSkincare.updateData(namaLama);  break;  case 4:  cout << "Masukkan Nama Produk: ";  cin.ignore();  getline(cin, namaProduk);  cout << "Masukkan Harga: ";  cin >> harga;  cout << "Masukkan Urutan Tertentu: ";  cin >> urutan;  tokoSkincare.tambahDataUrutanTertentu(namaProduk, harga, urutan);  break;  case 5:  cout << "Masukkan Urutan Tertentu yang ingin dihapus: ";  cin >> urutan;  tokoSkincare.hapusDataUrutanTertentu(urutan);  break;  case 6:  tokoSkincare.hapusSeluruhData();  break;  case 7:  tokoSkincare.tampilkanData();  break;  case 8:  cout << "Terima kasih telah menggunakan program ini." << endl;  break;  default:  cout << "Pilihan menu tidak valid." << endl;  break;  }  } while (pilihan != 8);  return 0;  } |

**Screenshot Program**

Graphical user interface, text, application

Description automatically generated

**Deskripsi Program**

1. Tambahkan produk Azarine dengan harga 65000 diantara Somethinc dan Skintific

Graphical user interface, text, application

Description automatically generated

1. Hapus produk wardah

Graphical user interface, text, application

Description automatically generated

1. Update produk Hanasui menjadi Cleora dengan harga 55.000

Graphical user interface, text, application

Description automatically generated

1. Tampilkan menu seperti dibawah ini

Toko Skincare Purwokerto

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

Pada menu 7, tampilan akhirnya akan menjadi seperti dibawah ini :

|  |  |
| --- | --- |
| Nama Produk | Harga |
| Originote | 60.000 |
| Somethinc | 150.000 |
| Azarine | 65.000 |
| Skintific | 100.000 |
| Cleora | 55.000 |

Graphical user interface, text, application

Description automatically generated

# BAB IV

## KESIMPULAN

Kesimpulan dari Single Linked List dan Double Linked List adalah sebagai berikut:

1. Single Linked List adalah struktur data linear yang terdiri dari sejumlah simpul (node) yang saling terhubung secara sekuensial, dimana setiap simpul memiliki satu pointer yang menunjuk ke simpul berikutnya. Sedangkan Double Linked List adalah struktur data linear yang mirip dengan Single Linked List, namun setiap simpul memiliki dua pointer, yaitu satu pointer yang menunjuk ke simpul sebelumnya (prev) dan satu pointer yang menunjuk ke simpul berikutnya (next).
2. Single Linked List memiliki kelebihan dalam penggunaan memori yang lebih efisien karena hanya memerlukan satu pointer untuk setiap simpul, sehingga memori yang digunakan lebih sedikit. Namun, Single Linked List memiliki keterbatasan dalam operasi reverse atau penelusuran ke simpul sebelumnya, karena tidak memiliki pointer prev. Sedangkan Double Linked List dapat mengatasi keterbatasan ini karena setiap simpul memiliki pointer prev dan next, sehingga operasi reverse atau penelusuran ke simpul sebelumnya bisa dilakukan dengan lebih efisien.
3. Salah satu kelebihan Double Linked List adalah kemampuannya untuk melakukan operasi insert atau delete pada simpul tengah (bukan di awal atau akhir) dengan efisien, karena hanya perlu mengatur pointer prev dan next pada simpul sekitarnya. Sedangkan pada Single Linked List, untuk melakukan operasi insert atau delete pada simpul tengah, perlu dilakukan penelusuran dari awal hingga simpul sebelum simpul yang ingin dimodifikasi, yang memerlukan waktu lebih lama.
4. Kedua jenis linked list ini cocok digunakan dalam situasi-situasi yang berbeda, tergantung pada kebutuhan dan kompleksitas permasalahan yang dihadapi. Single Linked List biasanya digunakan dalam kasus-kasus sederhana ketika penggunaan memori harus diatur dengan efisien, atau ketika operasi yang dilakukan hanya terbatas pada penambahan atau penghapusan simpul di awal atau akhir. Sementara Double Linked List digunakan dalam kasus-kasus yang memerlukan operasi reverse atau penelusuran ke simpul sebelumnya, atau ketika operasi insert atau delete perlu dilakukan pada simpul tengah dengan efisien.

Demikianlah kesimpulan dari Single Linked List dan Double Linked List. Kedua jenis linked list ini memiliki kelebihan dan keterbatasan masing-masing, dan pilihan penggunaan tergantung pada kebutuhan spesifik dari permasalahan yang dihadapi.