**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

Modul Praktikum 7



Disusun oleh:

Nama : Aryo Abi Putra

NIM : 2407111242

Dosen : T. Yudi Hadiwantara, S.Kom, M.Kom

Reni Fitri Yani, S.T., M.T

Asisten : Yudhitya M. Renandra

Muhammad Haikal Fikri

**PROGRAM STUDI TEKNIK INFORMATIKA FAKULTAS TEKNIK**  
**UNIVERSITAS RIAU**  
**GENAP 2024/2025**

**1. Tujuan Praktikum**

Praktikum ini bertujuan agar mahasiswa :

1. Memahami konsep membangun double linked list

2. Memahami konsep operasi menyisipkan sebagai simpul ujung(awal) dari double

linked list

3. Memahami konsep operasi membaca sebuah simpul tertentu

4. Memahami konsep operasi mencari sebuah simpul tertentu

5. Memahami konsep operasi menghapus simpul tertentu

6. Mengimplementasikan semua operasi double linked list dalam pemrograman

**2. Dasar Teori**

Double linked list dibentuk dengan menyusun sejumlah elemen sehingga

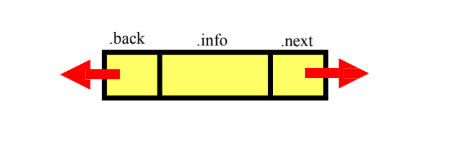
pointer next menunjuk ke elemen yang mengikutinya dan pointer back menunjuk ke

elemen yang mendahuluinya. Dalam gambar 7.1 ini diilustrasikan sebuah simpul

dalam double linked list. Info adalah data yang digunakan dalam simpul, back adalah

pointer yang menunjuk pada simpul sebelumnya, dan next adalah pointer yang

menunjuk pada simpul sesudahnya



Gambar 1.7 Ilustrasi sebuah simpul dalam Double linked list

**3. Pembahasan**

Praktikum ini dilaksanakan dengan melakukan percobaan dengan menerapkan double linked list

* **Percobaan** : Penerapan Double Linked List: Menyisipkan, Menampilkan, Mencari, dan Menghapus Simpul

*#include <iostream> // Library untuk input dan output*

*using namespace std; // Menggunakan namespace std agar tidak perlu menulis std:: secara eksplisit*

*struct Node {*

*int data; // Data yang disimpan di simpul*

*Node\* prev; // Pointer ke simpul sebelumnya*

*Node\* next; // Pointer ke simpul berikutnya*

*};*

*class DoubleLinkedList {*

*private:*

*Node\* head; // Pointer ke simpul pertama (head)*

*Node\* tail; // Pointer ke simpul terakhir (tail)*

*public:*

*DoubleLinkedList() : head(nullptr), tail(nullptr) {} // Konstruktor untuk menginisialisasi head dan tail sebagai nullptr*

*// 1. Menyisipkan simpul di awal*

*void insertFront(int data) {*

*Node\* newNode = new Node(); // Membuat simpul baru*

*newNode->data = data; // Mengisi data pada simpul baru*

*newNode->prev = nullptr; // Simpul baru tidak memiliki simpul sebelumnya*

*newNode->next = head; // Simpul baru menunjuk ke simpul pertama saat ini*

*if (head != nullptr) { // Jika list tidak kosong*

*head->prev = newNode; // Simpul pertama sebelumnya menunjuk ke simpul baru*

*} else {*

*tail = newNode; // Jika list kosong, tail juga menunjuk ke simpul baru*

*}*

*head = newNode; // Head diperbarui menjadi simpul baru*

*}*

*// 2. Menampilkan list*

*void display() {*

*Node\* current = head; // Memulai dari simpul pertama*

*if (current == nullptr) { // Jika list kosong*

*cout << "List kosong" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*cout << "Isi List: "; // Tampilkan pesan awal*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*cout << current->data << " "; // Tampilkan data simpul*

*current = current->next; // Pindah ke simpul berikutnya*

*}*

*cout << endl; // Baris baru setelah menampilkan semua data*

*}*

*// 3. Mencari simpul*

*void search(int data) {*

*Node\* current = head; // Memulai dari simpul pertama*

*int position = 1; // Posisi awal*

*bool found = false; // Penanda apakah data ditemukan*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*if (current->data == data) { // Jika data ditemukan*

*cout << "Data " << data << " ditemukan di posisi " << position << endl;*

*if (current == head) { // Jika data ada di simpul pertama*

*cout << "Ini adalah simpul pertama" << endl;*

*}*

*found = true; // Tandai bahwa data ditemukan*

*break; // Keluar dari loop*

*}*

*current = current->next; // Pindah ke simpul berikutnya*

*position++; // Tambahkan posisi*

*}*

*if (!found) { // Jika data tidak ditemukan*

*cout << "Data " << data << " tidak ditemukan" << endl;*

*}*

*}*

*// 4. Menghapus simpul*

*void deleteNode(int data) {*

*if (head == nullptr) { // Jika list kosong*

*cout << "List kosong" << endl;*

*return; // Keluar dari fungsi*

*}*

*Node\* current = head; // Memulai dari simpul pertama*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*if (current->data == data) { // Jika data ditemukan*

*// Jika simpul pertama*

*if (current == head) {*

*head = head->next; // Head diperbarui ke simpul berikutnya*

*if (head != nullptr) { // Jika list tidak kosong setelah penghapusan*

*head->prev = nullptr; // Simpul pertama baru tidak memiliki simpul sebelumnya*

*} else {*

*tail = nullptr; // Jika list kosong, tail juga null*

*}*

*}*

*// Jika simpul terakhir*

*else if (current == tail) {*

*tail = tail->prev; // Tail diperbarui ke simpul sebelumnya*

*tail->next = nullptr; // Simpul terakhir baru tidak memiliki simpul berikutnya*

*}*

*// Jika simpul di tengah*

*else {*

*current->prev->next = current->next; // Sambungkan simpul sebelumnya ke simpul berikutnya*

*current->next->prev = current->prev; // Sambungkan simpul berikutnya ke simpul sebelumnya*

*}*

*delete current; // Hapus simpul*

*cout << "Data " << data << " berhasil dihapus" << endl;*

*return; // Keluar dari fungsi*

*}*

*current = current->next; // Pindah ke simpul berikutnya*

*}*

*cout << "Data " << data << " tidak ditemukan" << endl; // Jika data tidak ditemukan*

*}*

*~DoubleLinkedList() { // Destruktor untuk membersihkan semua simpul saat objek dihancurkan*

*Node\* current = head; // Memulai dari simpul pertama*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*Node\* temp = current; // Simpan simpul saat ini*

*current = current->next; // Pindah ke simpul berikutnya*

*delete temp; // Hapus simpul saat ini*

*}*

*}*

*};*

*int main() {*

*DoubleLinkedList dll; // Membuat objek double linked list*

*int choice, data; // Variabel untuk pilihan menu dan data*

*do {*

*// Menampilkan menu operasi*

*cout << "\nMenu Operasi Double Linked List:\n";*

*cout << "1. Tambah di awal\n";*

*cout << "2. Tampilkan list\n";*

*cout << "3. Cari data\n";*

*cout << "4. Hapus data\n";*

*cout << "5. Keluar\n";*

*cout << "Pilihan: ";*

*cin >> choice; // Input pilihan dari pengguna*

*switch (choice) {*

*case 1: // Tambah di awal*

*cout << "Masukkan data: ";*

*cin >> data; // Input data*

*dll.insertFront(data); // Panggil fungsi insertFront*

*break;*

*case 2: // Tampilkan list*

*dll.display(); // Panggil fungsi display*

*break;*

*case 3: // Cari data*

*cout << "Cari data: ";*

*cin >> data; // Input data*

*dll.search(data); // Panggil fungsi search*

*break;*

*case 4: // Hapus data*

*cout << "Hapus data: ";*

*cin >> data; // Input data*

*dll.deleteNode(data); // Panggil fungsi deleteNode*

*break;*

*case 5: // Keluar*

*cout << "Program selesai" << endl;*

*break;*

*default: // Jika pilihan tidak valid*

*cout << "Pilihan tidak valid!" << endl;*

*}*

*} while (choice != 5); // Ulangi selama pilihan bukan 5 (keluar)*

*return 0; // Program selesai*

*}*

Program C++ ini mengimplementasikan struktur data double linked list yang memungkinkan penyisipan, pencarian, penampilan, dan penghapusan simpul secara dinamis. Struktur Node menyimpan data integer serta pointer ke simpul sebelumnya dan berikutnya untuk menjaga keterkaitan dua arah antar simpul. Kelas DoubleLinkedList mengelola daftar dengan pointer head dan tail, menyediakan fungsi untuk menambahkan simpul di awal daftar (insertFront), menampilkan seluruh isi daftar (display), mencari data tertentu dalam daftar (search), serta menghapus simpul berdasarkan nilai data (deleteNode). Program utama menyediakan antarmuka menu interaktif bagi pengguna untuk melakukan operasi-operasi tersebut secara berulang hingga memilih keluar. Destruktor kelas memastikan semua memori yang dialokasikan untuk simpul dibebaskan saat objek dihancurkan, sehingga mencegah kebocoran memori. Pendekatan ini memberikan fleksibilitas manipulasi elemen dalam list dengan efisiensi akses dua arah pada setiap node.

* **Latihan 1**

*#include <iostream>*

*using namespace std;*

*struct Node {*

*int data; // Data yang disimpan di simpul*

*Node\* prev; // Pointer ke simpul sebelumnya*

*Node\* next; // Pointer ke simpul berikutnya*

*};*

*class LifoLinkedList {*

*private:*

*Node\* top; // Pointer ke elemen teratas (terakhir dimasukkan)*

*Node\* bottom; // Pointer ke elemen terbawah (pertama dimasukkan)*

*public:*

*LifoLinkedList() : top(nullptr), bottom(nullptr) {} // Konstruktor untuk menginisialisasi stack kosong*

*// 1. Push: Menambahkan elemen di atas stack*

*void push(int data) {*

*Node\* newNode = new Node(); // Membuat simpul baru*

*newNode->data = data; // Mengisi data pada simpul baru*

*newNode->prev = top; // Simpul baru menunjuk ke elemen teratas saat ini*

*newNode->next = nullptr; // Simpul baru tidak memiliki elemen berikutnya*

*if (top == nullptr) { // Jika stack kosong*

*bottom = newNode; // Simpul baru menjadi elemen terbawah*

*} else {*

*top->next = newNode; // Elemen teratas saat ini menunjuk ke simpul baru*

*}*

*top = newNode; // Simpul baru menjadi elemen teratas*

*}*

*// 2. Pop: Menghapus elemen teratas*

*void pop() {*

*if (isEmpty()) { // Jika stack kosong*

*cout << "Stack kosong!" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* temp = top; // Simpan elemen teratas untuk dihapus*

*top = top->prev; // Perbarui elemen teratas menjadi elemen sebelumnya*

*if (top != nullptr) { // Jika stack tidak kosong setelah penghapusan*

*top->next = nullptr; // Elemen teratas baru tidak memiliki elemen berikutnya*

*} else { // Jika stack menjadi kosong*

*bottom = nullptr; // Elemen terbawah juga menjadi null*

*}*

*delete temp; // Hapus elemen teratas*

*cout << "Elemen teratas berhasil dihapus" << endl; // Tampilkan pesan*

*}*

*// 3. Peek: Melihat elemen teratas*

*void peek() {*

*if (isEmpty()) { // Jika stack kosong*

*cout << "Stack kosong!" << endl; // Tampilkan pesan*

*} else {*

*cout << "Elemen teratas: " << top->data << endl; // Tampilkan data elemen teratas*

*}*

*}*

*// 4. Menampilkan semua elemen*

*void display() {*

*if (isEmpty()) { // Jika stack kosong*

*cout << "Stack kosong!" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* current = bottom; // Mulai dari elemen terbawah*

*cout << "Isi Stack (dari bawah ke atas): "; // Tampilkan pesan awal*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*cout << current->data << " "; // Tampilkan data simpul*

*current = current->next; // Pindah ke simpul berikutnya*

*}*

*cout << endl; // Baris baru setelah menampilkan semua data*

*}*

*// 5. Mencari elemen*

*void search(int data) {*

*if (isEmpty()) { // Jika stack kosong*

*cout << "Stack kosong!" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* current = bottom; // Mulai dari elemen terbawah*

*int position = 1; // Posisi awal*

*bool found = false; // Penanda apakah data ditemukan*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*if (current->data == data) { // Jika data ditemukan*

*cout << "Data " << data << " ditemukan di posisi " << position; // Tampilkan posisi data*

*if (current == top) { // Jika data ada di elemen teratas*

*cout << " (elemen teratas)"; // Tampilkan keterangan tambahan*

*}*

*cout << endl; // Baris baru*

*found = true; // Tandai bahwa data ditemukan*

*}*

*current = current->next; // Pindah ke simpul berikutnya*

*position++; // Tambahkan posisi*

*}*

*if (!found) { // Jika data tidak ditemukan*

*cout << "Data " << data << " tidak ditemukan" << endl; // Tampilkan pesan*

*}*

*}*

*bool isEmpty() { // Fungsi untuk memeriksa apakah stack kosong*

*return top == nullptr; // Kembalikan true jika top adalah null*

*}*

*~LifoLinkedList() { // Destruktor untuk membersihkan semua simpul saat objek dihancurkan*

*while (!isEmpty()) { // Selama stack tidak kosong*

*pop(); // Hapus elemen teratas*

*}*

*}*

*};*

*int main() {*

*LifoLinkedList stack; // Membuat objek stack*

*int choice, data; // Variabel untuk pilihan menu dan data*

*do {*

*// Menampilkan menu operasi*

*cout << "\nMenu Operasi LIFO (Stack):\n";*

*cout << "1. Push (Tambah data)\n";*

*cout << "2. Pop (Hapus data)\n";*

*cout << "3. Peek (Lihat teratas)\n";*

*cout << "4. Tampilkan stack\n";*

*cout << "5. Cari data\n";*

*cout << "6. Keluar\n";*

*cout << "Pilihan: ";*

*cin >> choice; // Input pilihan dari pengguna*

*switch (choice) {*

*case 1: // Push (Tambah data)*

*cout << "Masukkan data: ";*

*cin >> data; // Input data*

*stack.push(data); // Panggil fungsi push*

*break;*

*case 2: // Pop (Hapus data)*

*stack.pop(); // Panggil fungsi pop*

*break;*

*case 3: // Peek (Lihat teratas)*

*stack.peek(); // Panggil fungsi peek*

*break;*

*case 4: // Tampilkan stack*

*stack.display(); // Panggil fungsi display*

*break;*

*case 5: // Cari data*

*cout << "Cari data: ";*

*cin >> data; // Input data*

*stack.search(data); // Panggil fungsi search*

*break;*

*case 6: // Keluar*

*cout << "Program selesai" << endl; // Tampilkan pesan keluar*

*break;*

*default: // Jika pilihan tidak valid*

*cout << "Pilihan tidak valid!" << endl; // Tampilkan pesan kesalahan*

*}*

*} while (choice != 6); // Ulangi selama pilihan bukan 6 (keluar)*

*return 0; // Program selesai*

*}*

Kode ini mengimplementasikan struktur data stack menggunakan linked list dengan prinsip LIFO (Last In First Out). Pertama, objek stack diinisialisasi dengan pointer top dan bottom yang menunjuk ke elemen teratas dan terbawah. Saat fungsi push dipanggil, sebuah node baru dibuat dan ditempatkan di atas stack, memperbarui pointer yang sesuai. Fungsi pop menghapus elemen teratas dari stack jika tidak kosong, sekaligus memperbarui pointer agar tetap konsisten. Fungsi peek menampilkan nilai elemen teratas tanpa menghapusnya. Fungsi display menelusuri seluruh elemen dari bawah ke atas untuk menampilkan isi stack secara berurutan. Selain itu, fungsi search mencari keberadaan data tertentu dalam stack dan melaporkan posisinya serta apakah data tersebut berada di puncak stack atau tidak. Proses ini terus berjalan melalui menu interaktif hingga pengguna memilih keluar, memastikan pengelolaan memori dilakukan dengan benar melalui destruktor yang membersihkan semua node saat program selesai. Hasil akhir adalah manipulasi dinamis data pada struktur stack sesuai prinsip LIFO menggunakan linked list sebagai media penyimpanan.

* **Latihan 2**

*#include <iostream> // Library untuk input dan output*

*using namespace std; // Menggunakan namespace std agar tidak perlu menulis std:: secara eksplisit*

*struct Node {*

*int data; // Data yang disimpan di simpul*

*Node\* next; // Pointer ke simpul berikutnya*

*};*

*class Queue {*

*private:*

*Node\* front; // Pointer ke elemen pertama*

*Node\* rear; // Pointer ke elemen terakhir*

*public:*

*Queue() : front(nullptr), rear(nullptr) {} // Konstruktor untuk menginisialisasi antrian kosong*

*// 1. Enqueue: Menambahkan elemen di akhir antrian*

*void enqueue(int data) {*

*Node\* newNode = new Node(); // Membuat simpul baru*

*newNode->data = data; // Mengisi data pada simpul baru*

*newNode->next = nullptr; // Simpul baru tidak memiliki simpul berikutnya*

*if (isEmpty()) { // Jika antrian kosong*

*front = rear = newNode; // Simpul baru menjadi elemen pertama dan terakhir*

*} else {*

*rear->next = newNode; // Elemen terakhir saat ini menunjuk ke simpul baru*

*rear = newNode; // Simpul baru menjadi elemen terakhir*

*}*

*cout << "Data " << data << " ditambahkan ke antrian" << endl; // Tampilkan pesan*

*}*

*// 2. Dequeue: Menghapus elemen dari depan antrian*

*void dequeue() {*

*if (isEmpty()) { // Jika antrian kosong*

*cout << "Antrian kosong!" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* temp = front; // Simpan elemen pertama untuk dihapus*

*front = front->next; // Perbarui elemen pertama menjadi elemen berikutnya*

*// Jika antrian menjadi kosong setelah dequeue*

*if (front == nullptr) {*

*rear = nullptr; // Elemen terakhir juga menjadi null*

*}*

*cout << "Data " << temp->data << " dihapus dari antrian" << endl; // Tampilkan pesan*

*delete temp; // Hapus elemen pertama*

*}*

*// 3. Menampilkan antrian*

*void display() {*

*if (isEmpty()) { // Jika antrian kosong*

*cout << "Antrian kosong" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* current = front; // Mulai dari elemen pertama*

*cout << "Isi Antrian (dari depan ke belakang): "; // Tampilkan pesan awal*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*cout << current->data << " "; // Tampilkan data simpul*

*current = current->next; // Pindah ke simpul berikutnya*

*}*

*cout << endl; // Baris baru setelah menampilkan semua data*

*}*

*// 4. Mencari elemen dalam antrian*

*void search(int data) {*

*if (isEmpty()) { // Jika antrian kosong*

*cout << "Antrian kosong!" << endl; // Tampilkan pesan*

*return; // Keluar dari fungsi*

*}*

*Node\* current = front; // Mulai dari elemen pertama*

*int position = 1; // Posisi awal*

*bool found = false; // Penanda apakah data ditemukan*

*while (current != nullptr) { // Iterasi selama simpul tidak null*

*if (current->data == data) { // Jika data ditemukan*

*cout << "Data " << data << " ditemukan di posisi " << position; // Tampilkan posisi data*

*if (current == front) { // Jika data ada di elemen pertama*

*cout << " (elemen terdepan)"; // Tampilkan keterangan tambahan*

*}*

*cout << endl; // Baris baru*

*found = true; // Tandai bahwa data ditemukan*

*}*

*current = current->next; // Pindah ke simpul berikutnya*

*position++; // Tambahkan posisi*

*}*

*if (!found) { // Jika data tidak ditemukan*

*cout << "Data " << data << " tidak ditemukan" << endl; // Tampilkan pesan*

*}*

*}*

*// Cek apakah antrian kosong*

*bool isEmpty() {*

*return front == nullptr; // Kembalikan true jika front adalah null*

*}*

*~Queue() { // Destruktor untuk membersihkan semua simpul saat objek dihancurkan*

*while (!isEmpty()) { // Selama antrian tidak kosong*

*dequeue(); // Hapus elemen pertama*

*}*

*}*

*};*

*int main() {*

*Queue q; // Membuat objek antrian*

*int choice, data; // Variabel untuk pilihan menu dan data*

*do {*

*// Menampilkan menu operasi*

*cout << "\nMenu Operasi FIFO (Queue):\n";*

*cout << "1. Enqueue (Tambah data)\n";*

*cout << "2. Dequeue (Hapus data)\n";*

*cout << "3. Tampilkan antrian\n";*

*cout << "4. Cari data\n";*

*cout << "5. Keluar\n";*

*cout << "Pilihan: ";*

*cin >> choice; // Input pilihan dari pengguna*

*switch (choice) {*

*case 1: // Enqueue (Tambah data)*

*cout << "Masukkan data: ";*

*cin >> data; // Input data*

*q.enqueue(data); // Panggil fungsi enqueue*

*break;*

*case 2: // Dequeue (Hapus data)*

*q.dequeue(); // Panggil fungsi dequeue*

*break;*

*case 3: // Tampilkan antrian*

*q.display(); // Panggil fungsi display*

*break;*

*case 4: // Cari data*

*cout << "Cari data: ";*

*cin >> data; // Input data*

*q.search(data); // Panggil fungsi search*

*break;*

*case 5: // Keluar*

*cout << "Program selesai" << endl; // Tampilkan pesan keluar*

*break;*

*default: // Jika pilihan tidak valid*

*cout << "Pilihan tidak valid!" << endl; // Tampilkan pesan kesalahan*

*}*

*} while (choice != 5); // Ulangi selama pilihan bukan 5 (keluar)*

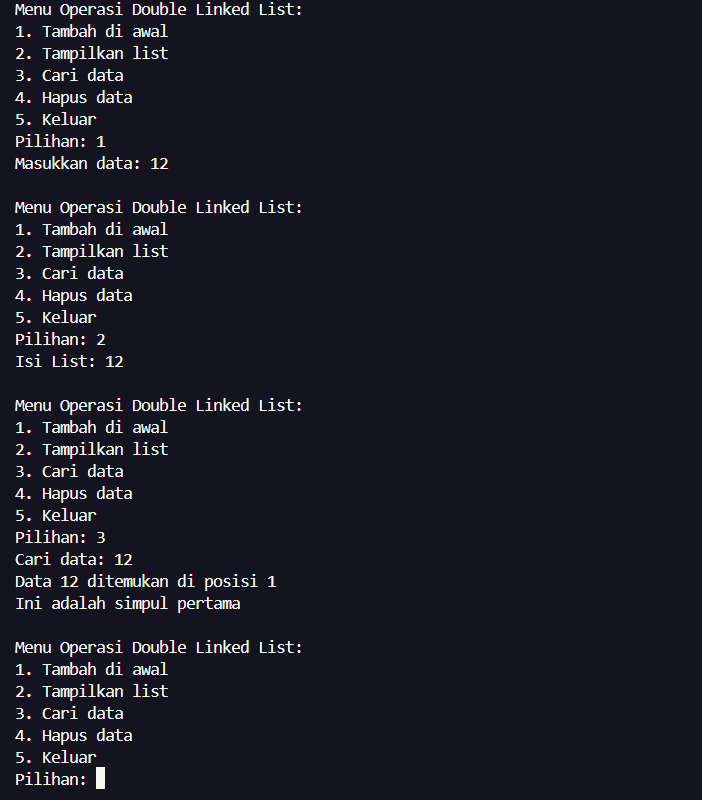
*return 0; // Program selesai*

*}*

Kode ini mengimplementasikan struktur data antrian (queue) menggunakan linked list dengan prinsip FIFO (First In First Out). Pertama, objek antrian diinisialisasi dengan pointer `front` dan `rear` yang menunjuk ke elemen terdepan dan terakhir. Saat fungsi `enqueue` dipanggil, sebuah node baru dibuat dan ditempatkan di belakang antrian, memperbarui pointer yang sesuai. Fungsi `dequeue` menghapus elemen terdepan dari antrian jika tidak kosong, sekaligus memperbarui pointer agar tetap konsisten. Fungsi `display` menelusuri seluruh elemen dari depan ke belakang untuk menampilkan isi antrian secara berurutan. Selain itu, fungsi `search` mencari keberadaan data tertentu dalam antrian dan melaporkan posisinya serta apakah data tersebut berada di posisi terdepan atau tidak. Proses ini terus berjalan melalui menu interaktif hingga pengguna memilih keluar, memastikan pengelolaan memori dilakukan dengan benar melalui destruktor yang membersihkan semua node saat program selesai. Hasil akhir adalah manipulasi dinamis data pada struktur queue sesuai prinsip FIFO menggunakan linked list sebagai media penyimpanan.

**4. Hasil**

**Hasil percobaan 1**

****

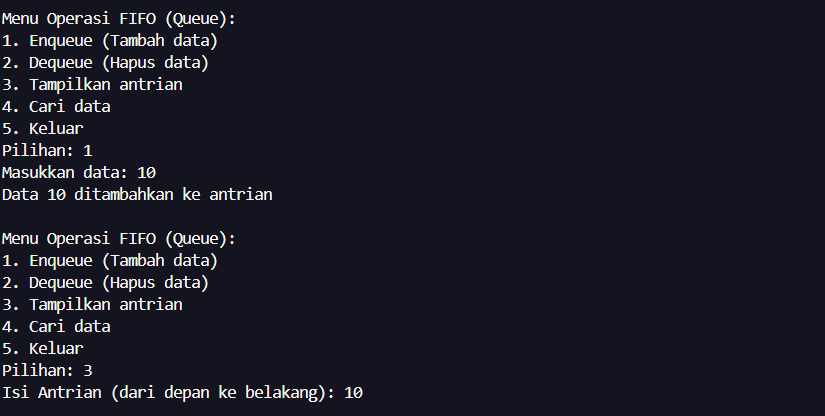
Output ini menunjukkan interaksi pengguna dengan program double linked list yang memungkinkan manipulasi data secara dinamis. Pertama, pengguna memilih opsi untuk menambahkan data di awal list dan memasukkan nilai 12. Kemudian, pengguna memilih untuk menampilkan isi list yang saat itu hanya berisi angka 12. Selanjutnya, pengguna mencari data 12 dalam list, dan program berhasil menemukan data tersebut pada posisi pertama serta menginformasikan bahwa itu adalah simpul pertama dalam daftar. Program menyediakan menu berulang sehingga pengguna dapat terus melakukan operasi seperti penambahan, pencarian, atau penghapusan hingga memilih keluar. Output ini menggambarkan bagaimana operasi dasar pada double linked list berjalan secara interaktif dan real-time sesuai input pengguna.

**Hasil latihan 1**

****

Output ini menunjukkan interaksi pengguna dengan program stack yang menerapkan prinsip LIFO (Last In First Out). Pertama, pengguna memilih opsi untuk menambahkan data ke stack dengan memasukkan nilai 10. Selanjutnya, pengguna memilih opsi untuk melihat elemen teratas (peek) pada stack, dan program menampilkan nilai 10 sebagai elemen teratas tersebut. Program menyediakan menu berulang sehingga pengguna dapat terus melakukan operasi seperti penambahan, penghapusan, pencarian, atau penampilan isi stack hingga memilih keluar. Output ini menggambarkan bagaimana operasi dasar pada struktur data stack berjalan secara interaktif sesuai input dari pengguna.

**Hasil Latihan 2**



Output ini menunjukkan interaksi pengguna dengan program antrian (queue) yang menerapkan prinsip FIFO (First In First Out). Pertama, pengguna memilih opsi untuk menambahkan data ke antrian dengan memasukkan nilai 10, dan program mengonfirmasi bahwa data tersebut berhasil ditambahkan. Selanjutnya, pengguna memilih opsi untuk menampilkan isi antrian, dan program menampilkan nilai 10 sebagai elemen tunggal dalam antrian dari depan ke belakang. Program menyediakan menu berulang sehingga pengguna dapat terus melakukan operasi seperti penambahan, penghapusan, pencarian, atau penampilan isi antrian hingga memilih keluar.

**5. Kesimpulan**

Praktikum ini berhasil mengimplementasikan dan memahami struktur data dinamis berupa double linked list, stack (LIFO), dan queue (FIFO) menggunakan linked list. Pada double linked list, dilakukan operasi penyisipan simpul di awal, penampilan isi daftar, pencarian data tertentu, serta penghapusan simpul dengan pengelolaan pointer dua arah yang efisien. Pada stack, operasi push menambahkan elemen di atas stack dan pop menghapus elemen teratas sesuai prinsip LIFO. Sedangkan pada queue, enqueue menambah elemen di belakang antrian dan dequeue menghapus elemen dari depan sesuai prinsip FIFO. Semua operasi dijalankan secara interaktif melalui menu sehingga memudahkan pemahaman konsep dasar struktur data tersebut sekaligus memastikan manajemen memori berjalan dengan baik melalui destruktor yang membersihkan node saat program selesai.

**6. Listing Program**

Percobaan:

#*include* <iostream> // *Library untuk input dan output*

*using* namespace std;// *Menggunakan namespace std agar tidak perlu menulis std:: secara eksplisit*

struct *Node* {

    int data;// *Data yang disimpan di simpul*

*Node*\* prev;// *Pointer ke simpul sebelumnya*

*Node*\* next;// *Pointer ke simpul berikutnya*

};

class *DoubleLinkedList* {

private:

*Node*\* head;// *Pointer ke simpul pertama (head)*

*Node*\* tail;// *Pointer ke simpul terakhir (tail)*

public:

*DoubleLinkedList*() : *head*(nullptr), *tail*(nullptr) {}// *Konstruktor untuk menginisialisasi head dan tail sebagai nullptr*

// *1. Menyisipkan simpul di awal*

    void *insertFront*(int data) {

*Node*\* newNode = **new** *Node*();// *Membuat simpul baru*

        newNode->data = data;// *Mengisi data pada simpul baru*

        newNode->prev = nullptr;// *Simpul baru tidak memiliki simpul sebelumnya*

        newNode->next = head;// *Simpul baru menunjuk ke simpul pertama saat ini*

*if* (head *!=* nullptr) {// *Jika list tidak kosong*

            head->prev = newNode;// *Simpul pertama sebelumnya menunjuk ke simpul baru*

        } *else* {

            tail = newNode;// *Jika list kosong, tail juga menunjuk ke simpul baru*

        }

        head = newNode;// *Head diperbarui menjadi simpul baru*

    }

// *2. Menampilkan list*

    void *display*() {

*Node*\* current = head;// *Memulai dari simpul pertama*

*if* (current *==* nullptr) {// *Jika list kosong*

            cout *<<* "List kosong" *<<* *endl*;// *Tampilkan pesan*

*return*;// *Keluar dari fungsi*

        }

        cout *<<* "Isi List: ";// *Tampilkan pesan awal*

*while* (current *!=* nullptr) {// *Iterasi selama simpul tidak null*

            cout *<<* current->data *<<* " ";// *Tampilkan data simpul*

            current = current->next;// *Pindah ke simpul berikutnya*

        }

        cout *<<* *endl*;// *Baris baru setelah menampilkan semua data*

    }

// *3. Mencari simpul*

    void *search*(int data) {

*Node*\* current = head;// *Memulai dari simpul pertama*

        int position = 1;// *Posisi awal*

        bool found = false;// *Penanda apakah data ditemukan*

*while* (current *!=* nullptr) {// *Iterasi selama simpul tidak null*

*if* (current->data *==* data) {// *Jika data ditemukan*

                cout *<<* "Data " *<<* data *<<* " ditemukan di posisi " *<<* position *<<* *endl*;

*if* (current *==* head) {// *Jika data ada di simpul pertama*

                    cout *<<* "Ini adalah simpul pertama" *<<* *endl*;

                }

                found = true;// *Tandai bahwa data ditemukan*

*break*;// *Keluar dari loop*

            }

            current = current->next;// *Pindah ke simpul berikutnya*

            position++;// *Tambahkan posisi*

        }

*if* (!found) {// *Jika data tidak ditemukan*

            cout *<<* "Data " *<<* data *<<* " tidak ditemukan" *<<* *endl*;

        }

    }

// *4. Menghapus simpul*

    void *deleteNode*(int data) {

*if* (head *==* nullptr) {// *Jika list kosong*

            cout *<<* "List kosong" *<<* *endl*;

*return*;// *Keluar dari fungsi*

        }

*Node*\* current = head;// *Memulai dari simpul pertama*

*while* (current *!=* nullptr) {// *Iterasi selama simpul tidak null*

*if* (current->data *==* data) {// *Jika data ditemukan*

// *Jika simpul pertama*

*if* (current *==* head) {

                    head = head->next;// *Head diperbarui ke simpul berikutnya*

*if* (head *!=* nullptr) {// *Jika list tidak kosong setelah penghapusan*

                        head->prev = nullptr;// *Simpul pertama baru tidak memiliki simpul sebelumnya*

                    } *else* {

                        tail = nullptr;// *Jika list kosong, tail juga null*

                    }

                }

// *Jika simpul terakhir*

*else* *if* (current *==* tail) {

                    tail = tail->prev;// *Tail diperbarui ke simpul sebelumnya*

                    tail->next = nullptr;// *Simpul terakhir baru tidak memiliki simpul berikutnya*

                }

// *Jika simpul di tengah*

*else* {

                    current->*prev*->next = current->next;// *Sambungkan simpul sebelumnya ke simpul berikutnya*

                    current->*next*->prev = current->prev;// *Sambungkan simpul berikutnya ke simpul sebelumnya*

                }

**delete** current;// *Hapus simpul*

                cout *<<* "Data " *<<* data *<<* " berhasil dihapus" *<<* *endl*;

*return*;// *Keluar dari fungsi*

            }

            current = current->next;// *Pindah ke simpul berikutnya*

        }

        cout *<<* "Data " *<<* data *<<* " tidak ditemukan" *<<* *endl*;// *Jika data tidak ditemukan*

    }

*~DoubleLinkedList*() {// *Destruktor untuk membersihkan semua simpul saat objek dihancurkan*

*Node*\* current = head;// *Memulai dari simpul pertama*

*while* (current *!=* nullptr) {// *Iterasi selama simpul tidak null*

*Node*\* temp = current;// *Simpan simpul saat ini*

            current = current->next;// *Pindah ke simpul berikutnya*

**delete** temp;// *Hapus simpul saat ini*

        }

    }

};

int *main*() {

*DoubleLinkedList* dll;// *Membuat objek double linked list*

    int choice, data;// *Variabel untuk pilihan menu dan data*

*do* {

// *Menampilkan menu operasi*

        cout *<<* "\nMenu Operasi Double Linked List:\n";

        cout *<<* "1. Tambah di awal\n";

        cout *<<* "2. Tampilkan list\n";

        cout *<<* "3. Cari data\n";

        cout *<<* "4. Hapus data\n";

        cout *<<* "5. Keluar\n";

        cout *<<* "Pilihan: ";

        cin *>>* choice;// *Input pilihan dari pengguna*

*switch* (choice) {

*case* 1:// *Tambah di awal*

                cout *<<* "Masukkan data: ";

                cin *>>* data;// *Input data*

                dll.*insertFront*(data);// *Panggil fungsi insertFront*

*break*;

*case* 2:// *Tampilkan list*

                dll.*display*();// *Panggil fungsi display*

*break*;

*case* 3:// *Cari data*

                cout *<<* "Cari data: ";

                cin *>>* data;// *Input data*

                dll.*search*(data);// *Panggil fungsi search*

*break*;

*case* 4:// *Hapus data*

                cout *<<* "Hapus data: ";

                cin *>>* data;// *Input data*

                dll.*deleteNode*(data);// *Panggil fungsi delessteNode*

*break*;

*case* 5:// *Keluar*

                cout *<<* "Program selesai" *<<* *endl*;

*break*;

*default*:// *Jika pilihan tidak valid*

                cout *<<* "Pilihan tidak valid!" *<<* *endl*;

        }

    } *while* (choice *!=* 5);// *Ulangi selama pilihan bukan 5 (keluar)*

*return* 0;// *Program selesai*}