

LAPORAN PRAKTIKUM

Modul 8

Queue



Disusun Oleh :

Satria Ariq Adelard

Dompas/2211104033SE 07 2

Asisten Praktikum :

Aldi Putra

Andini Nur Hidayah

Dosen Pengampu :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

1. Tujuan

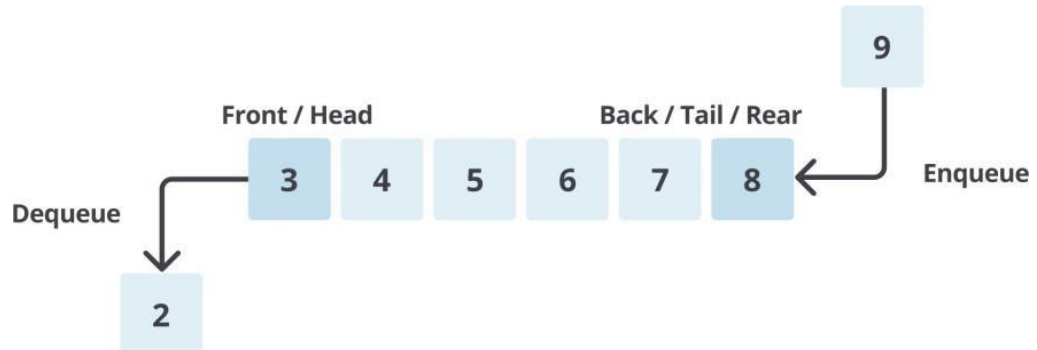
- Mahasiswa dapat mampu menjelaskan definisi dan konsep dari queue.
- Mahasiswa dapat mampu menerapkan operasi tambah, menghapus, pada queue.
- Mahasiswa dapat mampu menerapkan operasi tampil data pada queue.

2. Landasan Teori

a. Queue

Queue itu struktur data yang digunakan buat nyimpen data dengan cara FIFO (First-In First-Out). Jadi, data yang pertama kali masuk ke queue, bakal jadi yang pertama kali dikeluarkan juga. Konsepnya mirip kayak antrian di kehidupan sehari-hari, di mana orang yang datang duluan bakal dilayani duluan.

Queue bisa diimplementasikan dengan menggunakan array atau linked list. Struktur data queue punya dua pointer, yaitu front dan rear. Front itu pointer yang nunjuk ke elemen pertama di queue, sementara rear itu pointer yang nunjuk ke elemen terakhir di queue.



FIRST IN FIRST OUT (FIFO)

Perbedaan antara *stack* dan *queue* terletak pada aturan penambahan dan penghapusan elemen. Pada *stack*, operasi penambahan dan penghapusan elemen dilakukan di satu ujung yang disebut *top* (ujung atas). Elemen yang terakhir kali dimasukkan ke dalam *stack* akan berada di posisi paling atas dan akan menjadi elemen pertama yang dihapus. Sifat ini dikenal dengan istilah *LIFO* (Last In, First Out). Contoh analogi sederhana dari *stack* adalah tumpukan piring, di mana piring terakhir yang ditambahkan berada di posisi paling atas dan akan diambil atau dihapus terlebih dahulu.

Sebaliknya, pada *queue*, operasi penambahan dan penghapusan elemen dilakukan di dua ujung yang berbeda. Elemen baru ditambahkan di ujung belakang (*rear* atau *tail*), dan elemen dihapus dari ujung depan (*front* atau *head*). Proses ini mengikuti prinsip *FIFO* (First In, First Out), yang berarti elemen pertama yang dimasukkan ke dalam *queue* akan menjadi elemen pertama yang dikeluarkan. Dalam konteks *queue*, operasi penambahan elemen dikenal sebagai **Enqueue**, dan operasi penghapusan elemen disebut **Dequeue**.

Pada *Enqueue*, elemen ditambahkan di belakang *queue* setelah elemen terakhir yang ada, sementara pada *Dequeue*, elemen paling depan (*head*) dihapus, dan posisi *head* akan bergeser ke elemen berikutnya. Contoh penggunaan *queue* dalam kehidupan sehari-hari adalah antrean di kasir, di mana orang pertama yang datang adalah yang pertama dilayani.

Operasi pada Queue:

- enqueue() : menambahkan data ke dalam queue.
- dequeue() : mengeluarkan data dari queue.
- peek() : mengambil data dari queue tanpa menghapusnya.
- isEmpty() : mengecek apakah queue kosong atau tidak.
- isFull() : mengecek apakah queue penuh atau tidak.
- size() : menghitung jumlah elemen dalam queue.

3. Guided

a. Guided 1

Source Code

```
#include <iostream>
#define MAX 100

using namespace std;

class Queue
{
private:
    int front, rear;
    int arr[MAX];

public:
    Queue()
    {
```

```
        front = -1;
        rear = -1;
    }

    bool isFull()
    {
        return rear == MAX - 1;
    }

    bool isEmpty()
    {
        return front == -1 || front > rear;
    }

    void enqueue(int x)
    {
        if (isFull())
        {
            cout << "Queue Overflow\n";
            return;
        }
        if (front == -1)
            front = 0;
        arr[++rear] = x;
    }

    void dequeue()
    {
        if (isEmpty())
        {
            cout << "Queue Underflow\n";
            return;
        }
        front++;
    }

    int peek()
    {
        if (!isEmpty())
        {
            return arr[front];
        }
        cout << "Queue is empty\n";
        return -1;
    }

    void display()
    {
        if (isEmpty())
        {
            cout << "Queue is empty\n";
            return;
        }
    }
```

```

        for (int i = front; i <= rear; i++)
        {
            cout << arr[i] << " ";
        }
        cout << "\n";
    }
};

int main()
{
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << "Queue elements: ";
    q.display();

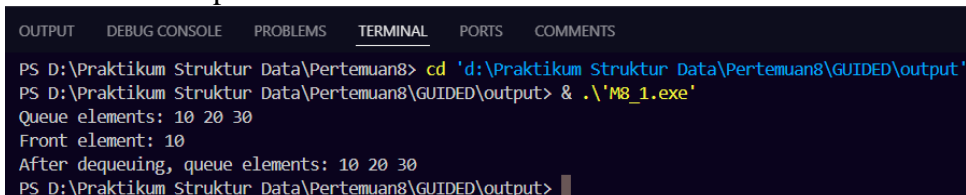
    cout << "Front element: " << q.peek() << "\n";

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}

```

Output



```

PS D:\Praktikum Struktur Data\Pertemuan8> cd 'd:\Praktikum Struktur Data\Pertemuan8\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> & .\M8_1.exe
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output>

```

Deskripsi

Kode di atas mengimplementasikan struktur data queue menggunakan array dengan kapasitas maksimum 100 elemen. Queue ini memiliki dua pointer, front dan rear, yang menunjukkan elemen pertama dan terakhir dalam antrian. Ada beberapa fungsi utama, seperti isFull() untuk memeriksa apakah queue sudah penuh, isEmpty() untuk memeriksa apakah queue kosong, enqueue(x) untuk menambahkan elemen ke belakang queue, dequeue() untuk menghapus elemen depan, dan peek() untuk melihat elemen paling depan tanpa menghapusnya. Fungsi display() digunakan untuk menampilkan semua elemen dalam queue. Program ini mendemonstrasikan cara kerja dasar queue dengan operasi enqueue, dequeue, peek, dan display.

b. Guided 2

Source Code

```
#include <iostream>

using namespace std;

// Node untuk setiap elemen Queue
class Node {
public:
    int data;        // Data elemen
    Node* next;      // Pointer ke node berikutnya

    // Konstruktor untuk Node
    Node(int value) {
        data = value;
        next = nullptr;
    }
};

// Kelas Queue menggunakan linked list
class Queue {
private:
    Node* front; // Pointer ke elemen depan Queue
    Node* rear;  // Pointer ke elemen belakang Queue

public:
    // Konstruktor Queue
    Queue() {
        front = rear = nullptr;
    }

    // Mengecek apakah Queue kosong
    bool isEmpty() {
        return front == nullptr;
    }

    // Menambahkan elemen ke Queue
    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {
            front = rear = newNode; // Jika Queue kosong
            return;
        }
        rear->next = newNode; // Tambahkan node baru ke belakang
        rear = newNode;      // Perbarui rear
    }

    // Menghapus elemen dari depan Queue
    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }
    }
};
```

```

    }
    Node* temp = front;        // Simpan node depan untuk dihapus
    front = front->next;       // Pindahkan front ke node berikutnya
    delete temp;              // Hapus node lama
    if (front == nullptr)     // Jika Queue kosong, rear juga harus null
        rear = nullptr;
}

// Mengembalikan elemen depan Queue tanpa menghapusnya
int peek() {
    if (!isEmpty()) {
        return front->data;
    }
    cout << "Queue is empty\n";
    return -1; // Nilai sentinel
}

// Menampilkan semua elemen di Queue
void display() {
    if (isEmpty()) {
        cout << "Queue is empty\n";
        return;
    }
    Node* current = front; // Mulai dari depan
    while (current) {      // Iterasi sampai akhir
        cout << current->data << " ";
        current = current->next;
    }
    cout << "\n";
}
};

// Fungsi utama untuk menguji Queue
int main() {
    Queue q;

    // Menambahkan elemen ke Queue
    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    // Menampilkan elemen di Queue
    cout << "Queue elements: ";
    q.display();

    // Menampilkan elemen depan
    cout << "Front element: " << q.peek() << "\n";

    // Menghapus elemen dari depan Queue
    q.dequeue();
    cout << "After dequeuing, queue elements: ";
    q.display();
}

```

```
    return 0;
}
```

Output

```
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> cd 'd:\Praktikum Struktur Data\Pertemuan8\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> & .\M8_2.exe
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> █
```

Deskripsi

Kode di atas mengimplementasikan struktur data queue menggunakan linked list. Setiap elemen dalam queue disimpan dalam node yang memiliki dua atribut: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya. Queue memiliki dua pointer, front dan rear, yang menunjuk ke elemen pertama dan terakhir. Fungsi-fungsi utama yang diimplementasikan adalah enqueue() untuk menambahkan elemen ke belakang queue, dequeue() untuk menghapus elemen dari depan queue, peek() untuk melihat elemen depan tanpa menghapusnya, dan display() untuk menampilkan seluruh elemen dalam queue. Program ini mendemonstrasikan penggunaan queue dengan operasi dasar tersebut dan menangani kondisi queue kosong.

c. Guided 3

Source Code

```
#include <iostream>

using namespace std;

const int maksimalQueue = 5; // Maksimal antrian
int front = 0;                // Penanda antrian
int back = 0;                 // Penanda
string queueTeller[5];        // Fungsi pengecekan

bool isFull()
{ // Pengecekan antrian penuh atau tidak
    if (back == maksimalQueue)
    {
        return true; // =1
    }
    else
    {
        return false;
    }
}
```



```

bool isEmpty()
{ // Antriannya kosong atau tidak
    if (back == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

void enqueueAntrian(string data)
{ // Fungsi menambahkan antrian
    if (isFull())
    {
        cout << "Antrian penuh" << endl;
    }
    else
    {
        if (isEmpty())
        { // Kondisi ketika queue kosong
            queueTeller[0] = data;
            front++;
            back++;
        }
        else
        { // Antrianya ada isi queueTeller[back] = data; back++;
        }
    }
}

void dequeueAntrian()
{ // Fungsi mengurangi antrian
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        for (int i = 0; i < back; i++)
        {
            queueTeller[i] = queueTeller[i + 1];
        }
        back--;
    }
}

int countQueue()
{ // Fungsi menghitung banyak antrian
    return back;
}

```

```

void clearQueue()
{ // Fungsi menghapus semua antrian
    if (isEmpty())
    {
        cout << "Antrian kosong" << endl;
    }
    else
    {
        for (int i = 0; i < back; i++)
        {
            queueTeller[i] = "";
        }
        back = 0;
        front = 0;
    }
}

void viewQueue()
{ // Fungsi melihat antrian
    cout << "Data antrian teller:" << endl;
    for (int i = 0; i < maksimalQueue; i++)
    {
        if (queueTeller[i] != "")
        {
            cout << i + 1 << ". " << queueTeller[i] <<

                endl;
        }
        else
        {
            cout << i + 1 << ". (kosong)" << endl;
        }
    }
}

int main()
{
    enqueueAntrian("Andi");

    enqueueAntrian("Maya");

    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    dequeueAntrian();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    clearQueue();
    viewQueue();
    cout << "Jumlah antrian = " << countQueue() << endl;

    return 0;
}

```

```
}
```

Output

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS
PS D:\Praktikum Struktur Data\Pertemuan8> cd 'd:\Praktikum Struktur Data\Pertemuan8\GUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> & .\'M8_3.exe'
Data antrian teller:
1. Andi
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 1
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
Antrian kosong
Data antrian teller:
1. (kosong)
2. (kosong)
3. (kosong)
4. (kosong)
5. (kosong)
Jumlah antrian = 0
PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> 
```

Deskripsi

Kode di atas mengimplementasikan antrian (queue) menggunakan array dengan kapasitas maksimal 5 elemen. Ada beberapa fungsi untuk mengelola antrian, seperti `isFull()` untuk mengecek apakah antrian sudah penuh, `isEmpty()` untuk mengecek apakah antrian kosong, `enqueueAntrian()` untuk menambahkan elemen ke antrian, dan `dequeueAntrian()` untuk menghapus elemen dari antrian. Fungsi lain termasuk `countQueue()` untuk menghitung jumlah elemen dalam antrian, `clearQueue()` untuk menghapus semua elemen antrian, dan `viewQueue()` untuk menampilkan kondisi antrian saat ini. Program ini menggambarkan operasi dasar antrian seperti menambah, menghapus, menghitung, dan melihat isi antrian, serta menangani kondisi penuh dan kosong.

4. Unguided

a. Soal 1

Source Code

```
#include <iostream>
```

```
using namespace std;

struct Node {
    int data;
    Node* next;
};

class Queue {
private:
    Node* front;
    Node* rear;

public:
    Queue() {
        front = nullptr;
        rear = nullptr;
    }

    bool isEmpty() {
        return front == nullptr;
    }

    void enqueue(int x) {
        Node* newNode = new Node();
        newNode->data = x;
        newNode->next = nullptr;

        if (rear == nullptr) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue Underflow\n";
            return;
        }

        Node* temp = front;
        front = front->next;
        if (front == nullptr) {
            rear = nullptr;
        }
        delete temp;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
    }
};
```

```

    }

    Node* temp = front;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
};

int main() {
    Queue q;

    q.enqueue(10);
    q.enqueue(20);
    q.enqueue(30);

    cout << "Queue elements: ";
    q.display();

    q.dequeue();

    cout << "After dequeuing, queue elements: ";
    q.display();

    return 0;
}

```

Output

```

PS D:\Praktikum Struktur Data\Pertemuan8\GUIDED\output> cd 'd:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output> & .\M8_1.exe
Queue elements: 10 20 30
After dequeuing, queue elements: 20 30
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output>

```

Deskripsi

Kode di atas mengimplementasikan antrian (queue) menggunakan linked list, di mana setiap elemen disimpan dalam node yang memiliki dua atribut: data untuk menyimpan nilai dan next yang menunjuk ke node berikutnya. Kelas Queue memiliki dua pointer, front dan rear, yang menunjuk ke elemen pertama dan terakhir. Fungsi enqueue() digunakan untuk menambahkan elemen baru ke antrian, dequeue() untuk menghapus elemen dari depan antrian, dan display() untuk menampilkan semua elemen dalam antrian. Program ini mendemonstrasikan penggunaan queue dengan operasi dasar, termasuk penambahan, penghapusan, dan penampilan elemen, serta menangani kondisi antrian kosong dan penuh.

b. Soal 2

Source Code

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

struct Node {
    string nama;
    string nim;
    Node* berikut;
};

class Queue {
private:
    Node* depan;
    Node* belakang;

public:
    Queue() {
        depan = nullptr;
        belakang = nullptr;
    }

    bool kosong() {
        return depan == nullptr;
    }

    void enqueue(string nama, string nim) {
        Node* nodeBaru = new Node();
        nodeBaru->nama = nama;
        nodeBaru->nim = nim;
        nodeBaru->berikut = nullptr;
```

```

        if (belakang == nullptr) {
            depan = belakang = nodeBaru;
        } else {
            belakang->berikut = nodeBaru;
            belakang = nodeBaru;
        }
    }

void dequeue() {
    if (kosong()) {
        cout << "Antrian kosong (Underflow)\n";
        return;
    }

    Node* sementara = depan;
    depan = depan->berikut;

    if (depan == nullptr) {
        belakang = nullptr;
    }

    delete sementara;
}

void tampilkan() {
    if (kosong()) {
        cout << "Antrian kosong\n";
        return;
    }

    Node* sementara = depan;
    while (sementara != nullptr) {
        cout << "Nama: " << sementara->nama << ", NIM: " << sementara-
>nim << "\n";
        sementara = sementara->berikut;
    }
}

};

int main() {
    Queue antrian;
    int jumlah;

    cout << "Masukkan jumlah mahasiswa yang ingin diinput: ";
    cin >> jumlah;

    for (int i = 0; i < jumlah; i++) {
        string nama, nim;
        cout << "Masukkan nama mahasiswa: ";
        cin >> ws;
        getline(cin, nama);
        cout << "Masukkan NIM mahasiswa: ";
        cin >> nim;
    }
}

```

```

        antrian.enqueue(nama, nim);
    }

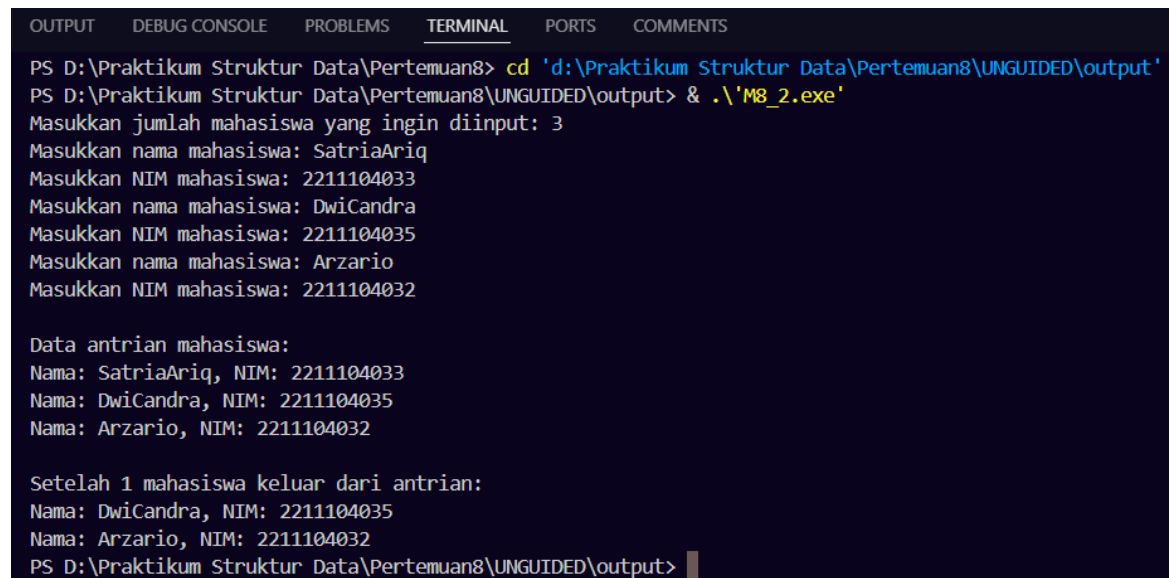
    cout << "\nData antrian mahasiswa:\n";
    antrian.tampilkan();

    antrian.dequeue();
    cout << "\nSetelah 1 mahasiswa keluar dari antrian:\n";
    antrian.tampilkan();

    return 0;
}

```

Output



```

OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS
PS D:\Praktikum Struktur Data\Pertemuan8> cd 'd:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output> & .\M8_2.exe'
Masukkan jumlah mahasiswa yang ingin diinput: 3
Masukkan nama mahasiswa: SatriaAriq
Masukkan NIM mahasiswa: 2211104033
Masukkan nama mahasiswa: DwiCandra
Masukkan NIM mahasiswa: 2211104035
Masukkan nama mahasiswa: Arzario
Masukkan NIM mahasiswa: 2211104032

Data antrian mahasiswa:
Nama: SatriaAriq, NIM: 2211104033
Nama: DwiCandra, NIM: 2211104035
Nama: Arzario, NIM: 2211104032

Setelah 1 mahasiswa keluar dari antrian:
Nama: DwiCandra, NIM: 2211104035
Nama: Arzario, NIM: 2211104032
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output>

```

Deskripsi

Kode di atas mengimplementasikan antrian (queue) dengan menggunakan struktur data linked list, yang menyimpan data mahasiswa berupa nama dan NIM. Kelas Queue memiliki dua pointer, depan dan belakang, yang menunjuk ke elemen pertama dan terakhir dalam antrian. Fungsi enqueue() digunakan untuk menambahkan mahasiswa ke dalam antrian, sedangkan dequeue() digunakan untuk menghapus mahasiswa dari antrian. Fungsi tampilkan() menampilkan seluruh antrian mahasiswa yang ada. Program ini meminta input dari pengguna untuk memasukkan sejumlah mahasiswa, menambahkannya ke dalam antrian, lalu menampilkan data antrian sebelum dan setelah satu elemen dikeluarkan.

c. Soal 3

Source Code

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

struct Node {
    string nama;
    string nim;
    Node* berikut;
};

class PriorityQueue {
private:
    Node* depan;

public:
    PriorityQueue() {
        depan = nullptr;
    }

    bool kosong() {
        return depan == nullptr;
    }

    void enqueue(string nama, string nim) {
        Node* nodeBaru = new Node();
        nodeBaru->nama = nama;
        nodeBaru->nim = nim;
        nodeBaru->berikut = nullptr;

        if (kosong() || nim < depan->nim) {
            nodeBaru->berikut = depan;
            depan = nodeBaru;
        } else {
            Node* sementara = depan;
            while (sementara->berikut != nullptr && sementara->berikut->nim
<= nim) {
                sementara = sementara->berikut;
            }
            nodeBaru->berikut = sementara->berikut;
            sementara->berikut = nodeBaru;
        }
    }

    void dequeue() {
        if (kosong()) {
            cout << "Antrian kosong (Underflow)\n";
            return;
        }

        Node* sementara = depan;
```

```

        depan = depan->berikut;
        delete sementara;
    }

    void tampilkan() {
        if (kosong()) {
            cout << "Antrian kosong\n";
            return;
        }

        Node* sementara = depan;
        while (sementara != nullptr) {
            cout << "Nama: " << sementara->nama << ", NIM: " << sementara-
>nim << "\n";
            sementara = sementara->berikut;
        }
    }
};

int main() {
    PriorityQueue antrian;
    int jumlah;

    cout << "Masukkan jumlah mahasiswa yang ingin diinput: ";
    cin >> jumlah;

    for (int i = 0; i < jumlah; i++) {
        string nama, nim;
        cout << "Masukkan nama mahasiswa: ";
        cin >> ws;
        getline(cin, nama);
        cout << "Masukkan NIM mahasiswa: ";
        cin >> nim;
        antrian.enqueue(nama, nim);
    }

    cout << "\nData antrian mahasiswa (berdasarkan prioritas NIM):\n";
    antrian.tampilkan();

    antrian.dequeue();
    cout << "\nSetelah 1 mahasiswa keluar dari antrian:\n";
    antrian.tampilkan();

    return 0;
}

```

Output

```
OUTPUT  DEBUG CONSOLE  PROBLEMS  TERMINAL  PORTS  COMMENTS
PS D:\Praktikum Struktur Data\Pertemuan8> cd 'd:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output'
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output> & .\M8_3.exe
Masukkan jumlah mahasiswa yang ingin diinput: 3
Masukkan nama mahasiswa: SatriaAriq
Masukkan NIM mahasiswa: 2211104033
Masukkan nama mahasiswa: DwiCandra
Masukkan NIM mahasiswa: 2211104035
Masukkan nama mahasiswa: Arzario
Masukkan NIM mahasiswa: 2211104032

Data antrian mahasiswa (berdasarkan prioritas NIM):
Nama: Arzario, NIM: 2211104032
Nama: SatriaAriq, NIM: 2211104033
Nama: DwiCandra, NIM: 2211104035

Setelah 1 mahasiswa keluar dari antrian:
Nama: SatriaAriq, NIM: 2211104033
Nama: DwiCandra, NIM: 2211104035
PS D:\Praktikum Struktur Data\Pertemuan8\UNGUIDED\output> █
```

Deskripsi

Kode di atas mengimplementasikan antrian prioritas (priority queue) dengan menggunakan struktur data linked list, yang menyimpan data mahasiswa berupa nama dan NIM. Kelas PriorityQueue memiliki metode enqueue() yang menambahkan mahasiswa ke dalam antrian dengan urutan berdasarkan NIM (NIM yang lebih kecil akan diprioritaskan untuk masuk ke depan antrian). Metode dequeue() menghapus mahasiswa dari depan antrian, dan tampilkan() menampilkan seluruh antrian. Program ini meminta input sejumlah mahasiswa, menambahkannya ke antrian berdasarkan prioritas NIM, lalu menampilkan data antrian sebelum dan setelah satu mahasiswa keluar.