

LAPORAN PRAKTIKUM

PERTEMUAN 8

QUEUE



Nama :

Alvin Bagus Firmansyah - 2311104070

Dosen :

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

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

1. Mahasiswa mampu menjelaskan definisi dan konsep dari queue
2. Mahasiswa mampu menerapkan operasi tambah, menghapus pada queue
3. Mahasiswa mampu menerapkan operasi tampil data pada queue

II. DASAR TEORI

Queue adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First-In First-Out). Data yang pertama dimasukkan ke dalam queue akan menjadi data yang pertama pula untuk dikeluarkan dari queue.

Implementasi queue dapat dilakukan dengan menggunakan array atau linked list. Struktur data queue terdiri dari dua pointer yaitu front dan rear. **Front/head** adalah pointer ke elemen pertama dalam queue dan **rear/tail/back** adalah pointer ke elemen terakhir dalam queue.



FIRST IN FIRST OUT (FIFO)

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

Di sisi lain, pada queue, elemen ditambahkan dan dihapus dari dua ujung yang berbeda.

Elemen baru ditambahkan di ujung belakang (rear atau tail), sementara elemen dihapus dari ujung depan (front atau head). Queue mengikuti prinsip FIFO (First In, First Out), yang berarti elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Proses penambahan elemen dalam queue disebut Enqueue, dan penghapusan elemen disebut Dequeue. Pada Enqueue, elemen ditambahkan di belakang antrian setelah elemen terakhir yang ada, sedangkan pada

Dequeue, elemen yang berada di depan (head) akan dihapus, dan posisi head akan bergeser ke elemen berikutnya. Sebagai contoh, antrian di kasir adalah penggunaan queue dalam kehidupan sehari-hari, di mana orang yang pertama 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.

III. GUIDE

1.Guided 1

Kode Program:

```
#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;
    }

```

Hasil Outputnya:

```

Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.

```

2. Guided 2

Kode Progam:

```

#include <iostream>

using namespace std;

class Node {
public:
    int data;
    Node* next;

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

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

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

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

    void enqueue(int x) {
        Node* newNode = new Node(x);
        if (isEmpty()) {

```

```

        front = rear = newNode;
        return;
    }
    rear->next = newNode;
    rear = newNode;
}

void dequeue() {
    if (isEmpty()) {
        cout << "Queue Underflow\n";
        return;
    }
    Node* temp = front;
    front = front->next;
    delete temp;
    if (front == nullptr)
        rear = nullptr;
}

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

void display() {
    if (isEmpty()) {
        cout << "Queue is empty\n";
        return;
    }
    Node* current = front;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    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";
}

```

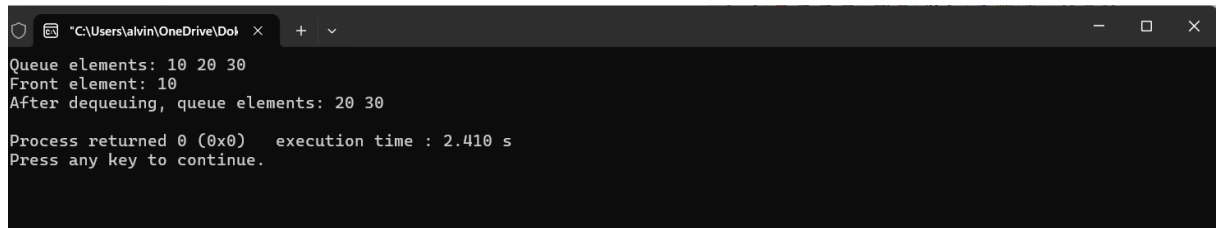
```

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

        return 0;
    }

```

Hasil Outputnya:



```

Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30

Process returned 0 (0x0)   execution time : 2.410 s
Press any key to continue.

```

3. Guided 3

Kode Progam:

```

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

// Mendefinisikan struktur node untuk data antrian mahasiswa
struct Mahasiswa {
    string nama;    // Nama mahasiswa
    string nim;     // NIM mahasiswa
    Mahasiswa* next; // Pointer ke node berikutnya
};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Mahasiswa* front; // Menunjuk ke elemen pertama dalam antrian
    Mahasiswa* back;  // Menunjuk ke elemen terakhir dalam antrian

public:
    // Konstruktor untuk inisialisasi antrian
    Queue() {
        front = nullptr;
        back = nullptr;
    }

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

    // Fungsi untuk menambahkan data ke antrian (enqueue) dengan prioritas
    // berdasarkan NIM
    void enqueue(string nama, string nim) {
        Mahasiswa* newNode = new Mahasiswa; // Membuat node baru
    }

```



```

newNode->nama = nama;
newNode->nim = nim;
newNode->next = nullptr;

if (isEmpty()) {
    // Jika antrian kosong, node baru menjadi front dan back
    front = newNode;
    back = newNode;
} else {
    // Jika antrian tidak kosong, kita cari posisi yang sesuai berdasarkan NIM
    if (stoi(nim) < stoi(front->nim)) {
        // Jika NIM lebih kecil dari NIM di depan, insert di depan
        newNode->next = front;
        front = newNode;
    } else {
        // Mencari posisi yang sesuai di antrian
        Mahasiswa* current = front;
        while (current->next != nullptr && stoi(nim) > stoi(current->next->nim)) {
            current = current->next;
        }
        // Menyisipkan node baru setelah node current
        newNode->next = current->next;
        current->next = newNode;

        // Jika node yang ditambahkan berada di belakang, update back
        if (newNode->next == nullptr) {
            back = newNode;
        }
    }
}

// Fungsi untuk mengeluarkan data dari antrian (dequeue)
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Mahasiswa* temp = front;
        front = front->next; // Menggeser front ke node berikutnya
        delete temp;        // Menghapus node lama
        if (front == nullptr) {
            back = nullptr; // Jika antrian kosong, update back menjadi null
        }
    }
}

// Fungsi untuk melihat data dalam antrian
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Mahasiswa* current = front;

```

```

        cout << "Data antrian mahasiswa:" << endl;
        int position = 1;
        while (current != nullptr) {
            cout << position << ". Nama: " << current->nama << ", NIM: " <<
            current->nim << endl;
            current = current->next;
            position++;
        }
    }
}

// Fungsi untuk menghitung jumlah data dalam antrian
int countQueue() {
    int count = 0;
    Mahasiswa* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}

// Fungsi untuk menghapus semua data dalam antrian
void clearQueue() {
    while (!isEmpty()) {
        dequeue(); // Menghapus data satu per satu
    }
    cout << "Antrian telah dikosongkan." << endl;
}

};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah input pilihan menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";

```

```

        getline(cin, nim);
        q.enqueue(nama, nim);
        break;
    case 2:
        q.dequeue();
        break;
    case 3:
        q.viewQueue();
        break;
    case 4:
        q.clearQueue();
        break;
    case 5:
        cout << "Jumlah antrian: " << q.countQueue() << endl;
        break;
    case 6:
        cout << "Terima kasih!" << endl;
        return 0;
    default:
        cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
    }
}
}

```

Hasil Outputnya:

```

C:\Users\alvin\OneDrive\Dot...
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
Process returned 0 (0x0)   execution time : 1.371 s
Press any key to continue.

```

IV. UNGUIDED

1. Ubahlah penerapan konsep queue pada bagian guided dari array menjadilinked list

Kode Progam:

```
#include <iostream>
```

```

#include <string>
using namespace std;

// Mendefinisikan struktur node untuk data antrian
struct Node {
    string data;    // Data yang disimpan (misalnya nama mahasiswa atau
                    // data lainnya)
    Node* next;    // Pointer ke node berikutnya
};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Node* front;    // Menunjuk ke elemen pertama dalam antrian
    Node* back;     // Menunjuk ke elemen terakhir dalam antrian

public:
    // Konstruktor untuk inisialisasi antrian
    Queue() {
        front = nullptr;
        back = nullptr;
    }

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

    // Fungsi untuk menambahkan data ke antrian (enqueue)
    void enqueue(string data) {
        Node* newNode = new Node; // Membuat node baru
        newNode->data = data;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode; // Jika antrian kosong, node baru menjadi front
            back = newNode;
        } else {
            back->next = newNode; // Menambahkan node baru di belakang
            back = newNode;      // Update back ke node baru
        }
    }

    // Fungsi untuk mengeluarkan data dari antrian (dequeue)
    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Node* temp = front;
            front = front->next; // Menggeser front ke node berikutnya
            delete temp;        // Menghapus node lama
            if (front == nullptr) {
                back = nullptr; // Jika antrian kosong, update back menjadi null
            }
        }
    }

    // Fungsi untuk melihat data dalam antrian
    void viewQueue() {

```

```

        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Node* current = front;
            cout << "Data antrian:" << endl;
            int position = 1;
            while (current != nullptr) {
                cout << position << ". " << current->data << endl;
                current = current->next;
                position++;
            }
        }
    }

    // Fungsi untuk menghitung jumlah data dalam antrian
    int countQueue() {
        int count = 0;
        Node* current = front;
        while (current != nullptr) {
            count++;
            current = current->next;
        }
        return count;
    }

    // Fungsi untuk menghapus semua data dalam antrian
    void clearQueue() {
        while (!isEmpty()) {
            dequeue(); // Menghapus data satu per satu
        }
        cout << "Antrian telah dikosongkan." << endl;
    }
};

int main() {
    Queue q;
    int pilihan;
    string data;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah input pilihan
        menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan data: ";
                getline(cin, data);
                q.enqueue(data);
                break;
            case 2:
                q.dequeue();

```

```

        break;
    case 3:
        q.viewQueue();
        break;
    case 4:
        q.clearQueue();
        break;
    case 5:
        cout << "Jumlah antrian: " << q.countQueue() << endl;
        break;
    case 6:
        cout << "Terima kasih!" << endl;
        return 0;
    default:
        cout << "Pilihan tidak valid! Silakan coba lagi." << endl;
    }
}
}

```

Hasil Ouputnya:

```

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan data: Alvin

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan data: Khenz

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 3
Data antrian:
1. Alvin
2. Khenz

```

2. Dari nomor 1 buatlah konsep antri dengan atribut Nama mahasiswa dan NIMMahasiswa

Kode Progam:

```

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

// Mendefinisikan struktur node untuk data antrian
mahasiswa
struct Mahasiswa {
    string nama;    // Nama mahasiswa
    string nim;     // NIM mahasiswa
    Mahasiswa* next; // Pointer ke node berikutnya
}

```

```

};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Mahasiswa* front; // Menunjuk ke elemen pertama dalam
                        antrian
    Mahasiswa* back;  // Menunjuk ke elemen terakhir dalam
                        antrian

public:
    // Konstruktor untuk inisialisasi antrian
    Queue() {
        front = nullptr;
        back = nullptr;
    }

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

    // Fungsi untuk menambahkan data ke antrian (enqueue)
    void enqueue(string nama, string nim) {
        Mahasiswa* newNode = new Mahasiswa; // Membuat
        node baru
        newNode->nama = nama;
        newNode->nim = nim;
        newNode->next = nullptr;

        if (isEmpty()) {
            front = newNode; // Jika antrian kosong, node baru
            menjadi front dan back
            back = newNode;
        } else {
            back->next = newNode; // Menambahkan node baru
            di belakang
            back = newNode;      // Update back ke node baru
        }
    }

    // Fungsi untuk mengeluarkan data dari antrian (dequeue)
    void dequeue() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {

```

```

        Mahasiswa* temp = front;
        front = front->next; // Menggeser front ke node
        berikutnya
        delete temp;        // Menghapus node lama
        if (front == nullptr) {
            back = nullptr; // Jika antrian kosong, update
            back menjadi null
        }
    }
}

// Fungsi untuk melihat data dalam antrian
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Mahasiswa* current = front;
        cout << "Data antrian mahasiswa:" << endl;
        int position = 1;
        while (current != nullptr) {
            cout << position << ". Nama: " << current->nama
            << ", NIM: " << current->nim << endl;
            current = current->next;
            position++;
        }
    }
}

// Fungsi untuk menghitung jumlah data dalam antrian
int countQueue() {
    int count = 0;
    Mahasiswa* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
    return count;
}

// Fungsi untuk menghapus semua data dalam antrian
void clearQueue() {
    while (!isEmpty()) {
        dequeue(); // Menghapus data satu per satu
    }
    cout << "Antrian telah dikosongkan." << endl;
}

```



```

};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah
        input pilihan menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";
                getline(cin, nim);
                q.enqueue(nama, nim);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:
                q.clearQueue();
                break;
            case 5:
                cout << "Jumlah antrian: " << q.countQueue() <<
endl;
                break;
            case 6:
                cout << "Terima kasih!" << endl;
                return 0;
            default:
                cout << "Pilihan tidak valid! Silakan coba lagi."

```

```

        << endl;
    }
}
}

```

Hasil Outputnya:

```

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Alvin
Masukkan NIM Mahasiswa: 12345

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Khenz
Masukkan NIM Mahasiswa: 6789

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 3
Data antrian mahasiswa:
1. Nama: Alvin, NIM: 12345
2. Nama: Khenz, NIM: 6789

```

3. Modifikasi program pada soal 1 sehingga mahasiswa dapat diprioritaskan berdasarkan NIM (NIM yang lebih kecil didahulukan pada saat output).

Kode Program:

```

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

// Mendefinisikan struktur node untuk data antrian
mahasiswa
struct Mahasiswa {
    string nama;    // Nama mahasiswa
    string nim;     // NIM mahasiswa
    Mahasiswa* next; // Pointer ke node berikutnya
};

// Mendefinisikan kelas Antrian dengan linked list
class Queue {
private:
    Mahasiswa* front; // Menunjuk ke elemen pertama dalam
    antrian
    Mahasiswa* back;  // Menunjuk ke elemen terakhir dalam
    antrian

public:
    // Konstruktor untuk inisialisasi antrian

```

```

Queue() {
    front = nullptr;
    back = nullptr;
}

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

// Fungsi untuk menambahkan data ke antrian (enqueue)
// dengan prioritas berdasarkan NIM
void enqueue(string nama, string nim) {
    Mahasiswa* newNode = new Mahasiswa; // Membuat
    node baru
    newNode->nama = nama;
    newNode->nim = nim;
    newNode->next = nullptr;

    if (isEmpty()) {
        // Jika antrian kosong, node baru menjadi front dan
        back
        front = newNode;
        back = newNode;
    } else {
        // Jika antrian tidak kosong, kita cari posisi yang
        sesuai berdasarkan NIM
        if (stoi(nim) < stoi(front->nim)) {
            // Jika NIM lebih kecil dari NIM di depan, insert di
            depan
            newNode->next = front;
            front = newNode;
        } else {
            // Mencari posisi yang sesuai di antrian
            Mahasiswa* current = front;
            while (current->next != nullptr && stoi(nim) >
            stoi(current->next->nim)) {
                current = current->next;
            }
            // Menyisipkan node baru setelah node current
            newNode->next = current->next;
            current->next = newNode;

            // Jika node yang ditambahkan berada di belakang,
            update back
            if (newNode->next == nullptr) {

```

```

        back = newNode;
    }
}
}

// Fungsi untuk mengeluarkan data dari antrian (dequeue)
void dequeue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Mahasiswa* temp = front;
        front = front->next; // Menggeser front ke node
        berikutnya
        delete temp;        // Menghapus node lama
        if (front == nullptr) {
            back = nullptr; // Jika antrian kosong, update
            back menjadi null
        }
    }
}

// Fungsi untuk melihat data dalam antrian
void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        Mahasiswa* current = front;
        cout << "Data antrian mahasiswa:" << endl;
        int position = 1;
        while (current != nullptr) {
            cout << position << ". Nama: " << current->nama
            << ", NIM: " << current->nim << endl;
            current = current->next;
            position++;
        }
    }
}

// Fungsi untuk menghitung jumlah data dalam antrian
int countQueue() {
    int count = 0;
    Mahasiswa* current = front;
    while (current != nullptr) {
        count++;
        current = current->next;
    }
}

```

```

    }
    return count;
}

// Fungsi untuk menghapus semua data dalam antrian
void clearQueue() {
    while (!isEmpty()) {
        dequeue(); // Menghapus data satu per satu
    }
    cout << "Antrian telah dikosongkan." << endl;
}
};

int main() {
    Queue q;
    int pilihan;
    string nama, nim;

    while (true) {
        cout << "\nMenu:\n";
        cout << "1. Tambah antrian\n";
        cout << "2. Hapus antrian\n";
        cout << "3. Lihat antrian\n";
        cout << "4. Hapus semua antrian\n";
        cout << "5. Hitung jumlah antrian\n";
        cout << "6. Keluar\n";
        cout << "Pilih menu: ";
        cin >> pilihan;
        cin.ignore(); // Untuk membersihkan newline setelah
        input pilihan menu

        switch (pilihan) {
            case 1:
                cout << "Masukkan Nama Mahasiswa: ";
                getline(cin, nama);
                cout << "Masukkan NIM Mahasiswa: ";
                getline(cin, nim);
                q.enqueue(nama, nim);
                break;
            case 2:
                q.dequeue();
                break;
            case 3:
                q.viewQueue();
                break;
            case 4:

```

```

        q.clearQueue();
        break;
    case 5:
        cout << "Jumlah antrian: " << q.countQueue() <<
endl;
        break;
    case 6:
        cout << "Terima kasih!" << endl;
        return 0;
    default:
        cout << "Pilihan tidak valid! Silakan coba lagi."
<< endl;
    }
}
}

```

Hasil Outputnya:

```

C:\Users\alvin\OneDrive\Doc...
Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Alvin
Masukkan NIM Mahasiswa: 123
Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Khenz
Masukkan NIM Mahasiswa: 456
Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 1
Masukkan Nama Mahasiswa: Daniel
Masukkan NIM Mahasiswa: 678
Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar

```

```

Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 3
Data antrian mahasiswa:
1. Nama: Alvin, NIM: 123
2. Nama: Khenz, NIM: 456
3. Nama: Daniel, NIM: 678
Menu:
1. Tambah antrian
2. Hapus antrian
3. Lihat antrian
4. Hapus semua antrian
5. Hitung jumlah antrian
6. Keluar
Pilih menu: 6
Terima kasih!
Process returned 0 (0x0)   execution time : 32.179 s
Press any key to continue.

```

V. KESIMPULAN

Implementasi queue dapat dilakukan dengan menggunakan array atau linked list. Masing-masing memiliki kelebihan dan kekurangan tersendiri. Penggunaan array lebih efisien dalam hal akses, namun memiliki keterbatasan dalam hal ukuran yang tetap. Sedangkan linked list lebih fleksibel dalam hal ukuran, namun kurang efisien dalam hal akses acak.

Dalam praktikum ini, telah dipelajari berbagai operasi pada queue, seperti enqueue (menambahkan elemen), dequeue (menghapus elemen), peek (melihat elemen terdepan), dan isEmpty (mengecek apakah queue kosong). Selain itu, telah diimplementasikan juga prioritas pada queue berdasarkan NIM mahasiswa, sehingga elemen dengan NIM terkecil akan diproses terlebih dahulu.