LAPORAN PRAKTIKUM
Modul 8
Queue

Disusun Oleh:
Zhafir Zaidan Avail
S1-SE-07-2

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

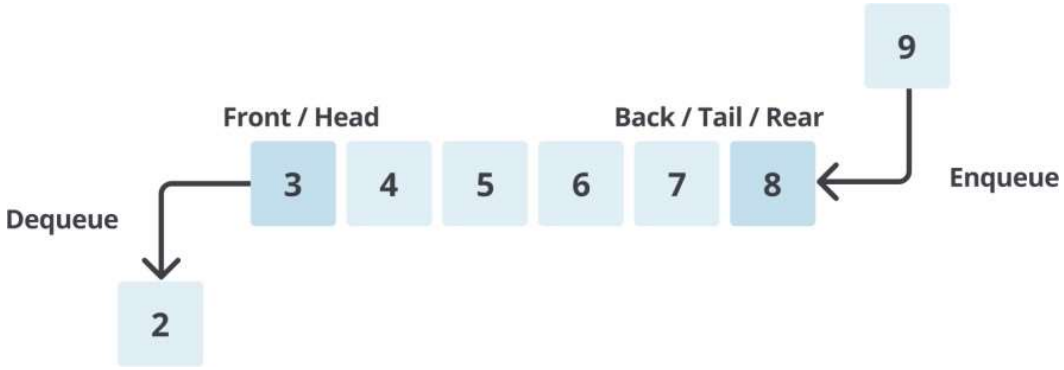
PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. **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
2. **Landasan 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 menjadidata yang pertama pula untuk dikeluarkan dari queue. Queue mirip dengan konsep **antrian** pada kehidupan sehari-hari, dimana konsumen yang datang lebih dulu akan dilayani terlebih dahulu.

    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)

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**

Guided1:

```
#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;
}
```

Gudied2

```
#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;
}
```

Guided3

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

const int maksimalQueue = 5;
```

```
int front = 0;
int back = 0;
string queueTeller[5];

bool isFull() {
    if (back == maksimalQueue) {
        return true;
    } else {
        return false;
    }
}
bool isEmpty() {
    if (back == 0) {
        return true;
    } else {
        return false;
    }
}
void enqueueAntrian(string data) {
    if (isFull()) {
        cout << "Antrian penuh" << endl;
    } else {
        if (isEmpty()) {
            queueTeller[0] = data;
            front++;
            back++;
        } else {
            queueTeller[back] = data;
            back++;
        }
    }
}
void dequeueAntrian() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        for (int i = 0; i < back; i++) {
            queueTeller[i] = queueTeller[i + 1];
        }
        back--;
    }
}
int countQueue() {
    return back;
}
void clearQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong" << endl;
    } else {
        for (int i = 0; i < back; i++) {
            queueTeller[i] = "";
        }
        back = 0;
        front = 0;
    }
}
void viewQueue() {
    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;
}
```

4. Unguided

Unguided1

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

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

    Node(string value) {
        data = value;
        next = nullptr;
    }
};
class Queue {
private:
    Node* front;
    Node* back;
    int size;
    const int maksimalQueue = 5;

public:
    Queue() {
        front = nullptr;
        back = nullptr;
        size = 0;
    }
    bool isFull() {
        return size == maksimalQueue;
    }
    bool isEmpty() {
        return size == 0;
    }
    void enqueueAntrian(string data) {
        if (isFull()) {
            cout << "Antrian penuh" << endl;
        } else {
            Node* newNode = new Node(data);
            if (isEmpty()) {
                front = back = newNode;
            } else {
                back->next = newNode;
                back = newNode;
            }
            size++;
        }
```

```
    }
    void dequeueAntrian() {
        if (isEmpty()) {
            cout << "Antrian kosong" << endl;
        } else {
            Node* temp = front;
            front = front->next;
            delete temp;
            size--;
            if (front == nullptr) {
                back = nullptr;
            }
        }
    }
    int countQueue() {
        return size;
    }
    void clearQueue() {
        while (!isEmpty()) {
            dequeueAntrian();
        }
        cout << "Antrian telah dikosongkan" << endl;
    }
    void viewQueue() {
        cout << "Data antrian teller:" << endl;
        if (isEmpty()) {
            cout << "(Kosong)" << endl;
        } else {
            Node* current = front;
            int i = 1;
            while (current != nullptr) {
                cout << i++ << ". " << current->data << endl;
                current = current->next;
            }
        }
    }
};

int main() {
    Queue q;

    q.enqueueAntrian("Andi");
    q.enqueueAntrian("Maya");

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

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

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

    return 0;
}
```

Output:

```
Data antrian teller:
1. Andi
2. Maya
Jumlah antrian = 2
Data antrian teller:
1. Maya
Jumlah antrian = 1
```

```
Antrian telah dikosongkan
Data antrian teller:
(Kosong)
Jumlah antrian = 0
```

Unguided2

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

class Node{
public:
    string nama;
    string nim;
    Node* next;
    Node(string namaMahasiswa, string nimMahasiswa){
        nama = namaMahasiswa;
        nim = nimMahasiswa;
        next = nullptr;
    }
};
class Queue{
private:
    Node* front;
    Node* back;
    int size;
    const int maksimalQueue = 5;

public:
    Queue(){
        front = nullptr;
        back = nullptr;
        size = 0;
    }
    bool isFull(){
        return size == maksimalQueue;
    }
    bool isEmpty(){
        return size == 0;
    }
    void enqueueAntrian(string namaMahasiswa, string nimMahasiswa) {
        if(isFull()){
            cout << "Antrian penuh" << endl;
        }else{
            Node* newNode = new Node(namaMahasiswa, nimMahasiswa);
            if (isEmpty()){
                front = back = newNode;
            }else{
                back->next = newNode;
                back = newNode;
            }
            size++;
        }
    }
    void dequeueAntrian(){
        if(isEmpty()){
            cout << "Antrian kosong" << endl;
        }else{
            Node* temp = front;
            front = front->next;
            delete temp;
            size--;
            if(front == nullptr){
                back = nullptr;
            }
        }
    }
```

```
    int countQueue(){
        return size;
    }
    void clearQueue(){
        while(!isEmpty()){
            dequeueAntrian();
        }
        cout<<"Antrian telah dikosongkan"<<endl;
    }
    void viewQueue(){
        cout<<"Data antrian mahasiswa:"<<endl;
        if(isEmpty()){
            cout<<"(Kosong)"<<endl;
        }else{
            Node* current = front;
            int i = 1;
            while(current != nullptr){
                cout<<i++<<". Nama: "<<current->nama
                     <<", NIM: "<<current->nim<<endl;
                current = current->next;
            }
        }
    }
};
int main(){
    Queue q;
    int pilihan;

    do{
        cout<<"\nMenu Antrian Mahasiswa:" <<endl;
        cout<<"1. Tambah mahasiswa ke antrian"<<endl;
        cout<<"2. Hapus mahasiswa dari antrian"<<endl;
        cout<<"3. Lihat semua antrian"<<endl;
        cout<<"4. Bersihkan antrian"<<endl;
        cout<<"5. Keluar"<<endl;
        cout<<"Pilihan: ";
        cin>>pilihan;

        switch(pilihan){
        case 1:{
            if(q.isFull()){
                cout<<"Antrian penuh, tidak dapat menambahkan data
baru."<<endl;
            }else{
                string nama, nim;
                cout<<"Masukkan nama mahasiswa: ";
                cin.ignore();
                getline(cin, nama);
                cout<<"Masukkan NIM mahasiswa: ";
                cin>>nim;
                q.enqueueAntrian(nama, nim);
            }
            break;
        }
        case 2:
            q.dequeueAntrian();
            break;
        case 3:
            q.viewQueue();
            break;
        case 4:
            q.clearQueue();
            break;
        case 5:
            cout<<"Keluar dari program."<<endl;
            break;
```

```
        default:
            cout<<"Pilihan tidak valid."<<endl;
        }
    }while(pilihan != 5);

    return 0;
}
```

Output

```
Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 1
Masukkan nama mahasiswa: Zhafir Zaidan Avail
Masukkan NIM mahasiswa: 2311104059

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 2

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
(Kosong)

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 1
Masukkan nama mahasiswa: Zhafir Zaiadan Avail
Masukkan NIM mahasiswa: 2311104059

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 4
Antrian telah dikosongkan

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 1
Masukkan nama mahasiswa: Zhafir Zaidan Avail
Masukkan NIM mahasiswa: 2311104059
```

```
Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
1. Nama: Zhafir Zaidan Avail, NIM: 2311104059

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 5
Keluar dari program.
```

Unguided3

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

class Node{
public:
    string nama;
    string nim;
    Node* next;
    Node(string namaMahasiswa, string nimMahasiswa){
        nama = namaMahasiswa;
        nim = nimMahasiswa;
        next = nullptr;
    }
};
class Queue{
private:
    Node* front;
    int size;
    const int maksimalQueue = 5;

public:
    Queue(){
        front = nullptr;
        size = 0;
    }
    bool isFull(){
        return size == maksimalQueue;
    }
    bool isEmpty(){
        return size == 0;
    }
    void enqueueAntrian(string namaMahasiswa, string nimMahasiswa){
        if(isFull()){
            cout<<"Antrian penuh"<<endl;
            return;
        }

        Node* newNode = new Node(namaMahasiswa, nimMahasiswa);

        if(isEmpty() || nimMahasiswa < front->nim){
            newNode->next = front;
            front = newNode;
        }else{
            Node* current = front;
```

```
            while(current->next != nullptr && current->next->nim <
nimMahasiswa){
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;
        }
        size++;
    }
    void dequeueAntrian(){
        if(isEmpty()){
            cout<<"Antrian kosong"<<endl;
            return;
        }
        Node* temp = front;
        front = front->next;
        delete temp;
        size--;
    }
    int countQueue(){
        return size;
    }
    void clearQueue(){
        while(!isEmpty()){
            dequeueAntrian();
        }
        cout<<"Antrian telah dikosongkan"<<endl;
    }
    void viewQueue(){
        cout<<"Data antrian mahasiswa:"<<endl;
        if(isEmpty()){
            cout<<"(Kosong)"<<endl;
        }else{
            Node* current = front;
            int i = 1;
            while(current != nullptr){
                cout<<i++<<". Nama: "<<current->nama
                    <<", NIM: "<<current->nim<<endl;
                current = current->next;
            }
        }
    }
};
int main(){
    Queue q;
    int pilihan;

    do{
        cout<<"\nMenu Antrian Mahasiswa:"<<endl;
        cout<<"1. Tambah mahasiswa ke antrian"<<endl;
        cout<<"2. Hapus mahasiswa dari antrian"<<endl;
        cout<<"3. Lihat semua antrian"<<endl;
        cout<<"4. Bersihkan antrian"<<endl;
        cout<<"5. Keluar"<<endl;
        cout<<"Pilihan: ";
        cin>>pilihan;

        switch(pilihan){
        case 1:{
            if(q.isFull()){
                cout<<"Antrian penuh, tidak dapat menambahkan data
baru."<<endl;
            }else{
                string nama, nim;
                cout<<"Masukkan nama mahasiswa: ";
                cin.ignore();
```

```
                getline(cin, nama);
                cout<<"Masukkan NIM mahasiswa: ";
                cin>>nim;
                q.enqueueAntrian(nama, nim);
            }
            break;
        }
        case 2:
            q.dequeueAntrian();
            break;
        case 3:
            q.viewQueue();
            break;
        case 4:
            q.clearQueue();
            break;
        case 5:
            cout<<"Keluar dari program."<<endl;
            break;
        default:
            cout<<"Pilihan tidak valid."<<endl;
        }
    }while(pilihan != 5);

    return 0;
}
```

Output

```
Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 1
Masukkan nama mahasiswa: Zhafir Zaidan Avail
Masukkan NIM mahasiswa: 2311104058

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 2

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan:
1
Masukkan nama mahasiswa: Zhafir Zaidan Avail
Masukkan NIM mahasiswa: 2311104059

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 3
Data antrian mahasiswa:
1. Nama: Zhafir Zaidan Avail, NIM: 2311104059
```

```
Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 4
Antrian telah dikosongkan

Menu Antrian Mahasiswa:
1. Tambah mahasiswa ke antrian
2. Hapus mahasiswa dari antrian
3. Lihat semua antrian
4. Bersihkan antrian
5. Keluar
Pilihan: 5
Keluar dari program.
```

5. Kesimpulan

Queue adalah struktur data yang bekerja berdasarkan prinsip FIFO (First In, First Out), mirip seperti antrian di dunia nyata. Data yang pertama masuk ke dalam queue akan menjadi yang pertama keluar. Operasi dasar pada queue adalah **enqueue** (menambahkan elemen ke belakang) dan **dequeue** (menghapus elemen dari depan). Queue dapat diimplementasikan menggunakan **array** atau **linked list**. Selain itu, queue juga memiliki operasi lain seperti **peek** (melihat elemen terdepan tanpa menghapus), **isEmpty** (mengecek apakah queue kosong), **isFull** (mengecek apakah queue penuh), dan **size** (menghitung jumlah elemen).

Konsep FIFO pada queue sangat berguna dalam berbagai aplikasi, seperti sistem antrian, pengelolaan tugas, dan simulasi. Dengan menggunakan queue, kita dapat mengatur urutan pemrosesan data dengan cara yang efisien dan sesuai dengan prinsip “siapa datang dulu, dia yang dilayani dulu”. Implementasi queue yang tepat dapat membantu meningkatkan kinerja dan efisiensi suatu sistem.