

**LAPORAN PRAKTIKUM  
STRUKTUR DATA  
Modul 8  
“QUEUE”**



**Disusun Oleh:  
MUHAMMAD RALFI - 2211104054  
SE-07-2**

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

**PROGRAM STUDI S1 SOFTWARE ENGINEERING  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY  
PURWOKERTO  
2024**

### 1. Tujuan

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

### 2. Landasan Teori

#### Queue

Queue atau antrian adalah struktur data yang digunakan untuk menyimpan data dengan metode FIFO (First In First Out) atau LILO (Last In Last Out). Contoh penerapan Queue pada kehidupan sehari-hari adalah seperti pada antrian pemesanan makanan, konsumen yang mengantri terlebih dahulu maka akan dilayani terlebih dahulu.

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

### 3. Guided

#### a. Guided 1

```
#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);

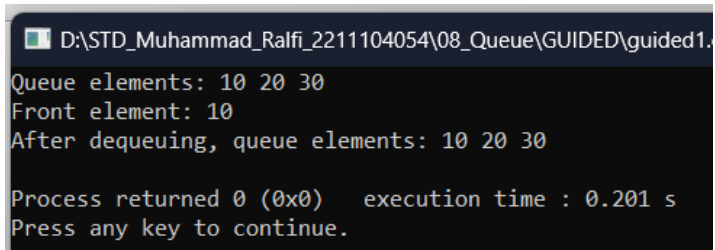
    cout << "Queue elements: ";
    q.display();

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

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

    return 0;
}
```

Output :



```
D:\STD_Muhammad_Ralfi_2211104054\08_Queue\GUIDED\guided1.
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 10 20 30

Process returned 0 (0x0)   execution time : 0.201 s
Press any key to continue.
```

b. Guided 2

```
#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
            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);

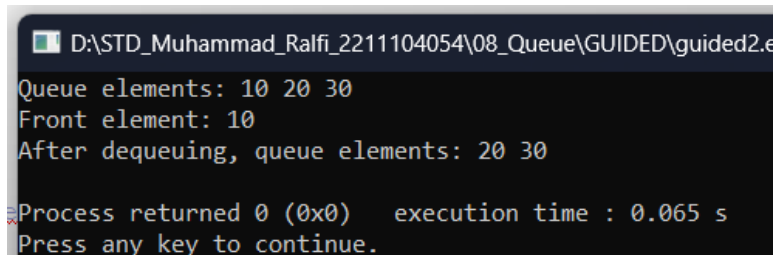
    // Menampilkan elemen di Queue
    cout << "Queue elements: ";
    q.display();

    // 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 :



```
D:\STD_Muhammad_Ralfi_2211104054\08_Queue\GUIDED\guided2.e
Queue elements: 10 20 30
Front element: 10
After dequeuing, queue elements: 20 30

Process returned 0 (0x0)   execution time : 0.065 s
Press any key to continue.
```

## c. Guided 3

```
#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 :

```

D:\STD_Muhammad_Ralfi_2211104054\08_Queue\GUIDED\guided3.
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 : 0.061 s
Press any key to continue.

```

#### 4. Unguided

##### a. Unguided 1

```

#include <iostream>
using namespace std;

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

Node* front = nullptr;
Node* rear = nullptr;

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

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

    if (isEmpty()) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << value << " berhasil ditambahkan ke antrian.\n";
}

void dequeue() {
    if (isEmpty()) {

```



```

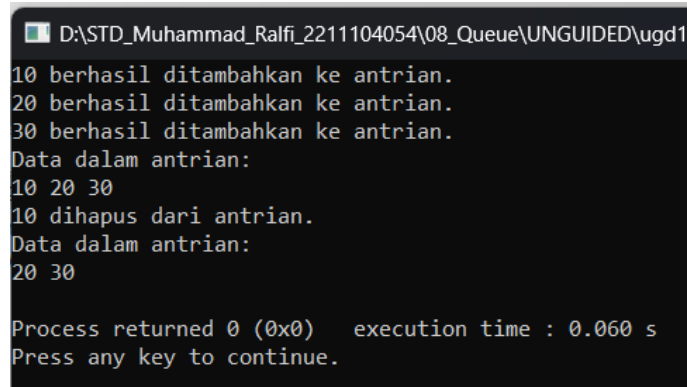
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    front = front->next;
    cout << temp->data << " dihapus dari antrian.\n";
    delete temp;
}

void viewQueue() {
    if (isEmpty()) {
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    cout << "Data dalam antrian:\n";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}

```

Output :



```

D:\STD_Muhammad_Ralfi_2211104054\08_Queue\UNGUIDED\ugd1
10 berhasil ditambahkan ke antrian.
20 berhasil ditambahkan ke antrian.
30 berhasil ditambahkan ke antrian.
Data dalam antrian:
10 20 30
10 dihapus dari antrian.
Data dalam antrian:
20 30

Process returned 0 (0x0)   execution time : 0.060 s
Press any key to continue.

```

#### b. Unguided 2

```

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

struct Node {
    string nama;
    string NIM;
    Node* next;
};

```

```
Node* front = nullptr;
Node* rear = nullptr;

void enqueue(string nama, string NIM) {
    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->NIM = NIM;
    newNode->next = nullptr;

    if (front == nullptr) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    cout << "Mahasiswa " << nama << " dengan NIM " << NIM << "
    berhasil ditambahkan.\n";
}

void dequeue() {
    if (front == nullptr) {
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp->NIM
    << " dikeluarkan dari antrian.\n";
    front = front->next;
    delete temp;
}

void viewQueue() {
    if (front == nullptr) {
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    cout << "Data dalam antrian:\n";
    while (temp != nullptr) {
        cout << "Nama: " << temp->nama << ", NIM: " << temp->NIM <<
        "\n";
        temp = temp->next;
    }
}

int main() {
    enqueue("Andi", "12345");
    enqueue("Budi", "67890");
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}
```

Output ;

```

D:\STD_Muhammad_Ralfi_2211104054\08_Queue\UNGUIDED\ugd2.exe
Mahasiswa Andi dengan NIM 12345 berhasil ditambahkan.
Mahasiswa Budi dengan NIM 67890 berhasil ditambahkan.
Data dalam antrian:
Nama: Andi, NIM: 12345
Nama: Budi, NIM: 67890
Mahasiswa Andi dengan NIM 12345 dikeluarkan dari antrian.
Data dalam antrian:
Nama: Budi, NIM: 67890

```

c. Unguided 3

```

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

struct Node {
    string nama;
    string NIM;
    Node* next;
};

Node* front = nullptr;

void enqueue(string nama, string NIM) {
    Node* newNode = new Node();
    newNode->nama = nama;
    newNode->NIM = NIM;
    newNode->next = nullptr;

    // Jika queue kosong atau NIM lebih kecil dari elemen pertama
    if (front == nullptr || NIM < front->NIM) {
        newNode->next = front;
        front = newNode;
    } else {
        // Menyisipkan elemen berdasarkan urutan NIM
        Node* temp = front;
        while (temp->next != nullptr && temp->next->NIM < NIM) {
            temp = temp->next;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }

    cout << "Mahasiswa " << nama << " dengan NIM " << NIM << "
    berhasil ditambahkan.\n";
}

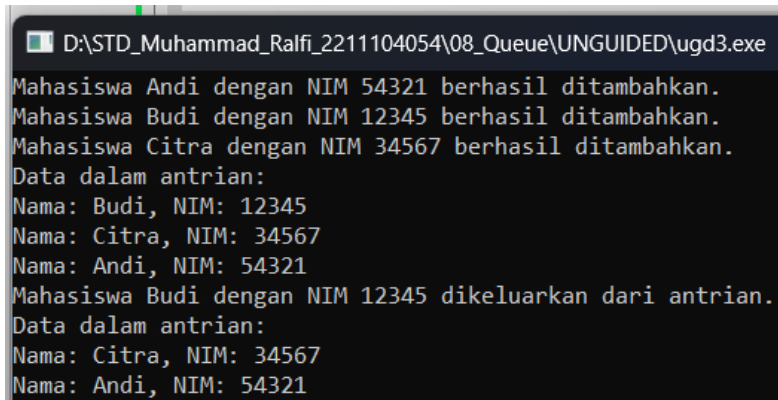
void dequeue() {
    if (front == nullptr) {
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    cout << "Mahasiswa " << temp->nama << " dengan NIM " << temp->NIM
    << " dikeluarkan dari antrian.\n";
    front = front->next;
    delete temp;
}

```

```
void viewQueue() {
    if (front == nullptr) {
        cout << "Antrian kosong.\n";
        return;
    }
    Node* temp = front;
    cout << "Data dalam antrian:\n";
    while (temp != nullptr) {
        cout << "Nama: " << temp->nama << ", NIM: " << temp->NIM <<
        "\n";
        temp = temp->next;
    }
}

int main() {
    enqueue("Andi", "54321");
    enqueue("Budi", "12345");
    enqueue("Citra", "34567");
    viewQueue();
    dequeue();
    viewQueue();
    return 0;
}
```

Output :



```
D:\STD_Muhammad_Ralfi_2211104054\08_Queue\UNGUIDED\ugd3.exe
Mahasiswa Andi dengan NIM 54321 berhasil ditambahkan.
Mahasiswa Budi dengan NIM 12345 berhasil ditambahkan.
Mahasiswa Citra dengan NIM 34567 berhasil ditambahkan.
Data dalam antrian:
Nama: Budi, NIM: 12345
Nama: Citra, NIM: 34567
Nama: Andi, NIM: 54321
Mahasiswa Budi dengan NIM 12345 dikeluarkan dari antrian.
Data dalam antrian:
Nama: Citra, NIM: 34567
Nama: Andi, NIM: 54321
```

## 5. Kesimpulan

Queue adalah struktur data linear yang mengikuti prinsip **FIFO (First-In, First-Out)**, di mana elemen pertama yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Struktur ini sering digunakan dalam berbagai aplikasi seperti manajemen antrian, pemrosesan data, dan sistem berbasis waktu nyata. Operasi dasar pada queue meliputi **enqueue** (menambahkan elemen), **dequeue** (menghapus elemen), dan **view** (melihat isi antrian). Implementasi queue dapat dilakukan menggunakan array atau linked list, dengan linked list memberikan fleksibilitas lebih untuk manipulasi data.