LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII
QUEUE

Disusun Oleh :
NAMA : Ridha Akifah
NIM : 103112400132

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue adalah struktur data linear yang bekerja dengan prinsip FIFO (First In First Out), yaitu elemen yang masuk pertama akan keluar terlebih dahulu, seperti antrian di loket atau kasir. Queue memiliki dua ujung, yaitu **head** sebagai posisi elemen yang akan dikeluarkan, dan **tail** sebagai posisi elemen baru yang masuk. Operasi utamanya meliputi **enqueue** (menambah elemen di bagian belakang), **dequeue** (menghapus elemen dari bagian depan), pengecekan **isEmpty**, **isFull**, serta menampilkan isi queue. Dalam modul 8, queue dijelaskan dalam dua bentuk representasi: menggunakan **pointer (linked list)** dan **array**, dengan tiga variasi mekanisme pada array, yaitu **head diam–tail bergerak**, **head & tail bergerak**, serta **circular queue** di mana head dan tail dapat berputar kembali ke awal. Struktur ini umum digunakan untuk sistem antrian, penjadwalan, buffer, dan proses yang membutuhkan urutan operasi yang teratur.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#define MAX_QUEUE 5

struct Queue {
    int info[MAX_QUEUE];
    int head;
    int tail;
    int count;
};

void createQueue(Queue &Q);

bool isEmpty(Queue Q);
```

```
bool isFull(Queue Q);

void enqueue(Queue &Q, int x);

int dequeue(Queue &Q);

void printInfo (Queue Q);

#endif
```

queue.cpp

```
//kode itu mengimplementasikan antrian (queue) berbasis array
dengan teknik circular buffer (penyimpanan melingkar)
#include "queue.h"
#include <iostream>

using namespace std; //Menggunakan namespace standar agar tidak
perlu menulis std::

// Definisi prosedur untuk membuat queue kosong
void createQueue(Queue &Q) {
    Q.head = 0;  // Set kepala ke indeks 0
    Q.tail = 0;  // Set ekor ke indeks 0
    Q.count = 0;  // Set jumlah elemen ke 0
}

// Definisi fungsi untuk mengecek apakah queue kosong
bool isEmpty(Queue Q) {
     return Q.count == 0; //Kembalikan true jika jumlah elemen
```

```
adalah 0
}

// Definisi fungsi untuk mengecek apakah queue penuh
bool isFull(Queue Q) {
     return Q.count == MAX_QUEUE; // Kembalikan true jika jumlah
elemen sama dengan maks
}

// Definisi prosedur untuk menambahkan elemen (enqueue)
void enqueue(Queue &Q, int x) {
    if (!isFull(Q)) { // Jika queue tidak penuh
          Q.info[Q.tail] = x; // Masukkan data (x) ke posisi ekor
(tail)
        // Pindahkan ekor secara circular (memutar)
        Q.tail = (Q.tail + 1) % MAX_QUEUE;
        Q.count++; //Tambah jumlah elemen
    } else { // Jika queue penuh
        cout << "Antrean Penuh!" << endl; //Tampilkan pesan error
    }
}

//Definisi fungsi untuk menghapus elemen (dequeue)
int dequeue(Queue &Q) {
    if (!isEmpty(Q)) { // Jika queue tidak kosong
          int x = Q.info[Q.head]; // Ambil data di posisi kepala
(head)
        //Pindahkan kepala secara circular (memutar)
        Q.head = (Q.head + 1) % MAX_QUEUE;
        Q.count--; // Kurangi jumlah elemen
```

```
        return x; // Kembalikan data yang diambil
    } else { // Jika queue kosong
            cout << "Antrean Kosong!" << endl; //Tampilkan pesan
error
        return -1; // Kembalikan nilai -1 sebagai tanda error
    }
}

// Definisi prosedur untuk menampilkan isi queue
void printInfo(Queue Q) {
    cout << "Isi Queue: [ "; // Tampilkan awalan
    if (!isEmpty(Q)) { // ika tidak kosong
        int i = Q.head; // Mulai dari kepala
        int n = 0; //Penghitung elemen yang sudah dicetak
        while (n < Q.count) { // Ulangi sebanyak jumlah elemen
            cout << Q.info[i] << " "; // Cetak info
            i = (i + 1) % MAX_QUEUE; // Geser 'i' secara circular
            n++; // Tambah penghitung
        }
    }
    cout << "]" << endl; // Tampilkan akhiran
}
```

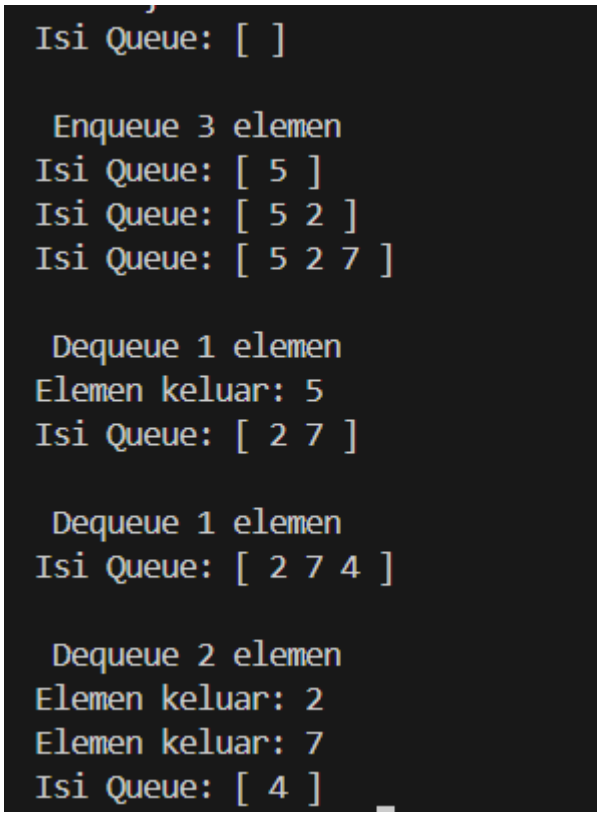
main.cpp

```
#include <iostream>
#include "queue.h"
#include "queue.cpp"

using namespace std;
```

```
int main ()
{
    Queue Q;

    createQueue(Q);
    printInfo(Q);

    cout << "\n Enqueue 3 elemen" << endl;
    enqueue(Q, 5);
    printInfo(Q);
    enqueue(Q, 2);
    printInfo(Q);
    enqueue(Q, 7);
    printInfo(Q);

    cout << "\n Dequeue 1 elemen" << endl;

    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q);

    cout << "\n Dequeue 1 elemen" << endl;
    enqueue(Q, 4);
    printInfo(Q);

    cout <<  "\n Dequeue 2 elemen" << endl;

    cout << "Elemen keluar: " << dequeue(Q) << endl;
```

```
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q);


    return 0;
}
```

Screenshoot Output



Deskripsi:

Program ini mengimplementasikan Circular Queue (Antrian Melingkar) menggunakan array dengan ukuran maksimum 5. Program ini memanfaatkan penanda head, tail, dan operasi modulo (%) untuk memastikan elemen dapat ditambahkan (enqueue) dan dihapus (dequeue) dengan mekanisme FIFO (First-In, First-Out) secara berputar tanpa membuang ruang array yang telah dikosongkan.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H
#define MAX_QUEUE 5
struct queue
{
    int info[MAX_QUEUE];
    int head;
    int tail;
    int count;
};
void createQueue(queue &Q);
bool isEmpty(queue Q);
bool isFull(queue Q);
void enqueue(queue &Q, int x);
int dequeue(queue &Q);
void printInfo(queue Q);
#endif
```

queue.cpp

```cpp
#include <iostream>
#include "queue.h"
using namespace std;
void createQueue(queue &Q)
{
    Q.head = -1;
    Q.tail = -1;
    Q.count = 0;
}
bool isEmpty(queue Q)
```

```
{
    return (Q.count == 0);
}
bool isFull(queue Q)
{
    return (Q.count == MAX_QUEUE);
}
void enqueue(queue &Q, int x)
{
    if (isFull(Q))
    {
        cout << "Queue penuh!\n";
        return;
    }
    if (isEmpty(Q))
    {
        Q.head = 0;
        Q.tail = 0;
    }
    else
    {
        Q.tail++;
    }
    Q.info[Q.tail] = x;
    Q.count++;
}
int dequeue(queue &Q)
{
    if (isEmpty(Q))
```

```
    {
        cout << "Queue kosong!\n";
        return -1;
    }
    int x = Q.info[Q.head];
    if (Q.head == Q.tail)
    {
        Q.head = -1;
        Q.tail = -1;
    }
    else
    {
        for (int i = Q.head; i < Q.tail; i++)
        {
            Q.info[i] = Q.info[i + 1];
        }
        Q.tail--;
    }
    Q.count--;
    return x;
}
void printInfo(queue Q)
{
    cout << Q.head << " - " << Q.tail << "    | ";
    if (isEmpty(Q))
    {
        cout << "empty queue\n";
        return;
    }
```

```
    for (int i = Q.head; i <= Q.tail; i++)
    {
        cout << Q.info[i] << " ";
    }
    cout << "\n";
}
```

main.cpp

```
#include <iostream>
#include "queue.h"
#include "queue.cpp"
using namespace std;

int main() {
    cout << "Hello world!" << endl;

    queue Q;
    createQueue(Q);

    cout << "----------------------" << endl;
    cout << " H - T    : Queue Info" << endl;
    cout << "----------------------" << endl;

    printInfo(Q);        // awal: empty
    enqueue(Q,5); printInfo(Q);
    enqueue(Q,2); printInfo(Q);
    enqueue(Q,7); printInfo(Q);
    dequeue(Q);   printInfo(Q);
    enqueue(Q,4); printInfo(Q);
```

```
    dequeue(Q);   printInfo(Q);
    dequeue(Q);   printInfo(Q);


      // sesuai modul, tampilkan sekali lagi untuk menunjukkan
antrean kosong akhir
    // (di modul ada baris -1 - -1 lagi)
    printInfo(Q);


    return 0;
}
```

Screenshoot Output

```
Hello world!
----------------------
 H - T    : Queue Info
----------------------
-1 - -1   | empty queue
 0 -  0   | 5
 0 -  1   | 5 2
 0 -  2   | 5 2 7
 0 -  1   | 2 7
 0 -  2   | 2 7 4
 0 -  1   | 7 4
 0 -  0   | 4
 0 -  0   | 4
```

Deskripsi:

Program ini mengimplementasikan Antrian Linier (ADT Queue) menggunakan array berukuran 5. Mekanismenya adalah FIFO (First-In, First-Out) di mana head selalu berada di index 0. Setiap operasi dequeue (penghapusan) memicu pergeseran (*shifting*) semua elemen yang tersisa ke kiri untuk mengisi kekosongan, sehingga mempertahankan head di awal array dan mengatur ulang tail.

Unguided 2

queue.h

tidak ada perubahan pada code

queue.cpp

hanya ada perubahan para void dequeue

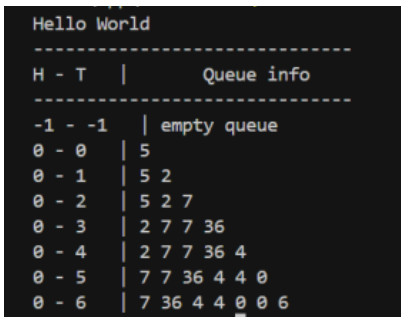
Sama dengan code sebelumnya hanya saja ada sedikit perubahan dari Q.head--; menjadi :

```
Q.head++;
```

Code main.cpp

tidak ada perubahan pada code

Screenshots Output



Deskripsi:

Program ini mengimplementasikan antrian linier (*linear queue*) di mana indeks head dan tail sama-sama bergerak maju (ke kanan) saat operasi enqueue dan dequeue dijalankan, tanpa memerlukan pergeseran elemen (*shifting*).

Unguided 3

queue.h

tidak ada perubahan pada code

queue.cpp

enqueue

Sama dengan code sebelumnya hanya saja ada sedikit perubahan dari Q.tail++; menjadi :

- dequeue

Sama dengan code sebelumnya hanya saja ada sedikit perubahan dari Q.head--; menjadi:

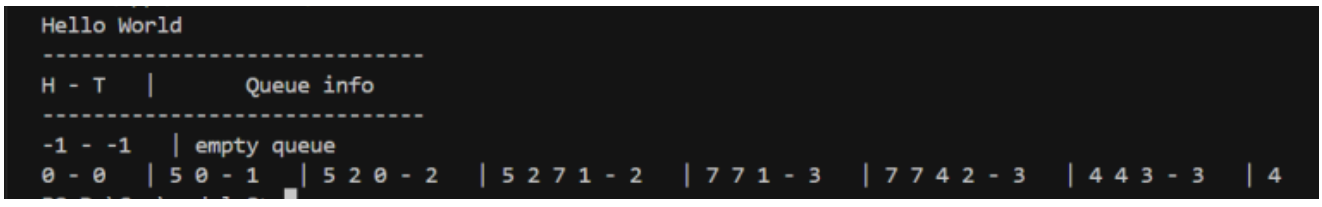
```
Q.head = (Q.head + 1) % MAX_QUEUE;
```

- printInfo

Sama dengan code sebelumnya hanya saja ada sedikit perubahan dari

```
for (int i = Q.head; i <= Q.tail; i++) {
cout << Q.info[i] << " ";
}
cout << "\n";
menjadi :
int i = Q.head;
for (int c = 0; c < Q.count; c++) {
cout << Q.info[i] << " ";
i = (i + 1) % MAX_QUEUE;
}
```

Screenshoot Output



Deskripsi

Implementasi antrian ini menggunakan konsep Circular Queue (Antrian Melingkar) di mana head dan tail bergerak memutar dalam array. Metode ini memungkinkan pemanfaatan penuh ruang array dan mencegah kondisi *false full* (antrian penuh palsu).

D. Kesimpulan

Modul 8 menyimpulkan bahwa queue adalah struktur data berprinsip FIFO yang memproses data berdasarkan urutan kedatangan, dengan dua operasi utama yaitu enqueue untuk menambah elemen dan dequeue untuk menghapus elemen. Modul ini menjelaskan bahwa queue dapat dibuat menggunakan pointer maupun array, serta memperkenalkan tiga cara kerja queue berbasis array, yaitu head diam tail bergerak, head dan tail bergerak, serta circular queue yang lebih efisien. Inti dari modul ini adalah memahami cara kerja antrian agar dapat diterapkan dengan tepat pada berbagai kebutuhan pemrosesan data yang memerlukan urutan yang teratur.

E. Referensi

https://en.wikipedia.org/wiki/Operators_in_C_and_C%2B%2B

https://learn.microsoft.com/id-id/cpp/cpp/references-cpp?view=msvc-170

https://medium.com/@mohamedeissabay/understanding-c-a-deep-dive-into-core-concepts-48a560679cdd