LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII
QUEUE

Disusun Oleh :
NAMA : RISKY CAHAYU
NIM : 103112430121

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Struktur data queue (antrian) adalah salah satu struktur data linear yang bekerja dengan prinsip FIFO (First In First Out), yaitu data yang pertama masuk akan menjadi data yang pertama keluar. Queue memiliki dua penunjuk utama, yaitu head (penunjuk elemen terdepan yang akan dikeluarkan) dan tail (penunjuk elemen terakhir tempat data baru ditambahkan). Pada implementasi circular queue (antrian melingkar), indeks array dihubungkan secara melingkar sehingga ketika posisi tail mencapai indeks terakhir, ia dapat kembali ke indeks awal selama masih ada ruang kosong, yang ditandai dengan perhitungan (tail + 1) % MAX. Teknik ini bertujuan untuk mengoptimalkan penggunaan memori dan menghindari pemborosan ruang yang sering terjadi pada queue biasa (linear queue). Operasi utama dalam queue yaitu enqueue (menambah data di bagian belakang) dan dequeue (menghapus data di bagian depan), serta operasi tambahan seperti pengecekan isEmpty (queue kosong) dan isFull (queue penuh). Pada circular queue, head akan bergerak saat proses dequeue sedangkan tail akan bergerak saat proses enqueue, sementara data tidak perlu digeser posisinya di dalam array, sehingga lebih efisien dalam penggunaan waktu maupun memori. Struktur data ini banyak digunakan dalam berbagai sistem seperti manajemen antrian proses pada sistem operasi, buffer data, dan sistem antrian pelayanan.

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

Guided 1

Kode main.cpp

```cpp
// Risky Cahayu
// 103112430121

#include <iostream>  // Menyertakan library untuk input/output
#include "queue.h"  // Menyertakan file header queue kita

using namespace std; // Menggunakan namespace standar

// Fungsi utama program
int main() {
    Queue Q; // Deklarasikan variabel queue bernama Q

    createQueue(Q); // Panggil prosedur untuk inisialisasi queue
    printInfo(Q);   //Tampilkan isi queue (seharusnya kosong)

    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;
    // Hapus 1 elemen dan tampilkan nilainya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q); // Tampilkan isi queue setelah dequeue

    cout << "\n Enqueue 1 Elemen" << endl;
    enqueue(Q, 4);
    printInfo(Q);

    cout << "\nDequeue 2 Elemen" << endl;
    // Hapus 1 elemen dan tampilkan nilainya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    // Hapus 1 elemen lagi dan tampilkan nilainya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q); // Tampilkan isi queue

    return 0;
}
```

Kode queue.cpp

```
// Risky Cahayu
// 103112430121

#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
}
```

```cpp

// 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
}
```

Kode queue.h

```cpp
// Risky Cahayu
// 103112430121

#ifndef QUEUE_H  // Jika QUEUE_H belum didefinisikan
```

```h
#define QUEUE_H // Maka definisikan QUEUE_H untuk mencegah inklusi ganda

#define MAX_QUEUE 5 // Menentukan ukuran maksimal antrean

// Mendefinisikan struktur (tipe data) untuk Queue
struct Queue {
    int info[MAX_QUEUE]; // Array untuk menyimpan data antrean
    int head;            //  Penanda untuk elemen paling depan (kepala)
    int tail;            //  Penanda untuk elemen paling belakang (ekor)
    int count;           // Penghitung jumlah elemen saat ini
};

//  Prosedur untuk membuat queue kosong
void createQueue(Queue &Q);

// Fungsi untuk mengecek apakah queue kosong
bool isEmpty(Queue Q);

// Fungsi untuk mengecek apakah queue penuh
bool isFull(Queue Q);

// Prosedur untuk menambahkan elemen ke queue (enqueue)
void enqueue(Queue &Q, int x);

// Fungsi untuk menghapus dan mengembalikan elemen dari queue (dequeue)
int dequeue(Queue &Q);

// Prosedur untuk menampilkan semua isi queue
void printInfo(Queue Q);

#endif
```

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\Guided\Modul 8> g++ main.cpp queue.cpp -o main
>> .\main
>>
Isi Queue: [ ]

 Enqueue 3 Elemen
Isi Queue: [ 5 ]
Isi Queue: [ 5 2 ]
Isi Queue: [ 5 2 7 ]

 Dequeue 1 Elemen
Elemen keluar: 5
Isi Queue: [ 2 7 ]

 Enqueue 1 Elemen
Isi Queue: [ 2 7 4 ]

Dequeue 2 Elemen
Elemen keluar: 2
Elemen keluar: 7
Isi Queue: [ 4 ]
PS C:\Users\ASUS\Videos\strukdat\Guided\Modul 8>
```

Deskripsi:

Kode program tersebut mengimplementasikan struktur data Queue (antrian) statis dengan metode circular array (antrian melingkar) menggunakan bahasa C++. Program terdiri dari tiga bagian utama, yaitu main.cpp, queue.cpp, dan queue.h. Pada file header queue.h didefinisikan struktur Queue yang berisi array info sebagai tempat penyimpanan elemen, serta tiga variabel penting yaitu head (penunjuk elemen paling depan), tail (penunjuk posisi penambahan elemen berikutnya), dan count (jumlah elemen saat ini), dengan kapasitas maksimal MAX_QUEUE = 5. File implementasi berisi fungsi createQueue untuk menginisialisasi queue menjadi kosong, isEmpty untuk mengecek apakah queue kosong, isFull untuk mengecek apakah queue penuh, enqueue untuk menambahkan data ke posisi tail dengan pergeseran melingkar menggunakan operasi modulo, dequeue untuk menghapus dan mengembalikan data dari posisi head, serta printInfo untuk menampilkan isi queue dari depan ke belakang. Pada fungsi main, program mendemonstrasikan penggunaan queue dengan mengosongkan queue terlebih dahulu, menambahkan tiga elemen (5, 2, 7), menghapus satu elemen, menambahkan satu elemen lagi (4), lalu menghapus dua elemen berikutnya, sambil menampilkan isi queue setelah setiap operasi. Dengan desain circular ini, posisi head dan tail akan kembali ke awal array ketika mencapai batas akhir, sehingga penggunaan memori menjadi efisien dan queue dapat digunakan secara optimal tanpa perlu memindahkan elemen secara manual.

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

Unguided 1 #Alternatif 1 (head diam, tail bergerak)

Kode main.cpp

```
// Risky Cahayu
```

```cpp
// 103112430121

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

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

    Queue Q;
    createQueue(Q);

    cout << "----------------------" << endl;
    cout << "H - T \t | Queue info" << endl;
    cout << "----------------------" << endl;

    printInfo(Q);

    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);

    return 0;
}
```

Kode queue.h

```cpp
// Risky Cahayu
// 103112430121

#ifndef QUEUE_H
#define QUEUE_H

#define MAX 5
typedef int infotype;

struct Queue {
    infotype info[MAX];
    int head;
    int tail;
};

void createQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
```

```
bool isFullQueue(Queue Q);
void enqueue(Queue &Q, infotype x);
infotype dequeue(Queue &Q);
void printInfo(Queue Q);

#endif
```

Kode queue.cpp

```
// Risky Cahayu
// 103112430121

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

void createQueue(Queue &Q) {
    Q.head = -1;
    Q.tail = -1;
}

bool isEmptyQueue(Queue Q) {
    return (Q.head == -1 && Q.tail == -1);
}

bool isFullQueue(Queue Q) {
    return (Q.tail == MAX - 1);   // tail mentok di akhir array
}

// Enqueue (head tetap, tail maju)
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
        return;
    }

    if (isEmptyQueue(Q)) {
        Q.head = Q.tail = 0;
    } else {
        Q.tail++;
    }

    Q.info[Q.tail] = x;
}

// Dequeue (head maju, tanpa shifting)
infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
```

```
        cout << "Queue kosong!" << endl;
        return -1;
    }

    infotype x = Q.info[Q.head];

    // Jika hanya 1 elemen
    if (Q.head == Q.tail) {
        Q.head = Q.tail = -1;
    } else {
        Q.head++;     // inilah yang membuat "head bergerak"
    }

    return x;
}

// Print sesuai format
void printInfo(Queue Q) {
    cout << Q.head << "   " << Q.tail << "   ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

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

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\Alternatif 1> g++ main.cpp queue.cpp -o main
>> .\main
>>
Hello World!
---------------------
H - T    | Queue info
---------------------
-1   -1   empty queue
0   0   5
0   1   5 2
0   2   5 2 7
1   2   2 7
1   3   2 7 4
2   3   7 4
3   3   4
PS C:\Users\ASUS\Videos\strukdat\Alternatif 1>
```

Deskripsi:

Kode “Alternatif 1” ini mengimplementasikan struktur data queue (antrian) linear menggunakan array statis dengan pendekatan sederhana tanpa mekanisme circular (melingkar) atau shifting data. Queue direpresentasikan oleh struktur Queue yang memiliki array info[MAX] berukuran 5 untuk menyimpan elemen, serta dua indeks yaitu head (penunjuk elemen terdepan) dan tail (penunjuk elemen terakhir), yang keduanya diinisialisasi dengan nilai -1 pada fungsi createQueue sebagai tanda bahwa queue masih kosong. Fungsi isEmptyQueue memeriksa kondisi kosong berdasarkan head dan tail yang bernilai -1, sedangkan isFullQueue menyatakan queue penuh jika posisi tail sudah mencapai indeks terakhir array. Prosedur enqueue menambahkan elemen di posisi tail dengan cara menaikkan nilai tail (atau mengatur head dan tail ke 0 jika sebelumnya kosong), tanpa memindahkan elemen lain. Fungsi dequeue menghapus elemen di posisi head dan menaikkan nilai head untuk menunjuk elemen berikutnya; jika hanya tersisa satu elemen, maka queue dikosongkan kembali dengan head dan tail diatur ke -1. Prosedur printInfo menampilkan posisi head dan tail serta seluruh isi queue dari indeks head sampai tail. Di dalam main, program mendemonstrasikan operasi antrean secara berurutan dengan menambahkan (5, 2, 7), menghapus satu elemen, menambahkan (4), lalu menghapus dua elemen lagi sambil menampilkan perubahan posisi head–tail dan isi queue setiap langkah, sehingga terlihat jelas bagaimana mekanisme kerja antrian linear tanpa pergeseran data (shifting).

Unguided 2 #Alternatif 2 (head bergerak, tail bergerak)

Kode queue.cpp

```cpp
// Risky Cahayu
// 103112430121

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

void createQueue(Queue &Q) {
    Q.head = -1;
    Q.tail = -1;
}

bool isEmptyQueue(Queue Q) {
    return (Q.head == -1 && Q.tail == -1);
}

bool isFullQueue(Queue Q) {
    return (Q.tail == MAX - 1);
}

// ==============================
//  ENQUEUE Alternatif 2
```

```
// ================================
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
        return;
    }

    if (isEmptyQueue(Q)) {
        Q.head = Q.tail = 0;
    } else {
        Q.tail++;
    }

    Q.info[Q.tail] = x;
}

// ================================
//  DEQUEUE Alternatif 2 (head bergerak)
// ================================
infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
    }

    infotype x = Q.info[Q.head];

    if (Q.head == Q.tail) {
        Q.head = Q.tail = -1;
    } else {
        Q.head++;
    }

    return x;
}

// ================================
//  PRINT (tetap sama seperti Alternatif 1)
// ================================
void printInfo(Queue Q) {
    cout << Q.head << "   " << Q.tail << "   ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    for (int i = Q.head; i <= Q.tail; i++) {
```

```
        cout << Q.info[i] << " ";
    }
    cout << endl;
}
```

Screenshots Output

Deskripsi:

Kode Alternatif 2 ini mengimplementasikan struktur data queue (antrian) linear berbasis array statis dengan ukuran maksimum MAX = 5, menggunakan dua penunjuk utama yaitu head sebagai penunjuk elemen terdepan dan tail sebagai penunjuk elemen terakhir dalam antrean. Queue dianggap kosong ketika head dan tail bernilai -1, yang diatur pada fungsi createQueue. Fungsi isEmptyQueue digunakan untuk memastikan apakah antrean kosong, sedangkan isFullQueue memeriksa apakah tail sudah mencapai batas akhir array, menandakan antrean penuh. Proses enqueue dilakukan dengan menaikkan posisi tail untuk menambahkan elemen baru di belakang antrean (sedangkan head tetap/diam), dan jika antrean masih kosong maka head dan tail sama-sama diatur ke indeks awal yaitu 0. Proses dequeue dilakukan dengan mengambil elemen pada posisi head kemudian menggeser posisi head ke depan (menunjuk elemen berikutnya) tanpa melakukan pergeseran data di dalam array, dan apabila hanya tersisa satu elemen maka queue dikosongkan kembali dengan mengatur head dan tail menjadi -1. Prosedur printInfo digunakan untuk menampilkan nilai head, tail, dan semua data di dalam queue dari indeks head hingga tail. Pada fungsi main, seluruh operasi queue ditunjukkan secara bertahap mulai dari antrian kosong, dilakukan beberapa kali enqueue (5, 2, 7), dequeue (1 kali), enqueue lagi (4), dan dequeue dua kali, sehingga memvisualisasikan bagaimana pergerakan tail saat menambah data dan pergerakan head saat menghapus data pada antrian linear tanpa mekanisme circular maupun shifting elemen.

Unguided 3 #Alternatif 3 (head dan tail berputar)

Kode queue.cpp

```cpp
// Risky Cahayu
// 103112430121

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

void createQueue(Queue &Q) {
    Q.head = -1;
    Q.tail = -1;
    Q.count = 0;
}

bool isEmptyQueue(Queue Q) {
    return (Q.count == 0);
}

bool isFullQueue(Queue Q) {
    return (Q.count == MAX);
}

// ================================
// ENQUEUE — Circular Queue
// ================================
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
        return;
    }

    if (isEmptyQueue(Q)) {
        Q.head = Q.tail = 0;
    }
    else {
        Q.tail = (Q.tail + 1) % MAX;
    }

    Q.info[Q.tail] = x;
    Q.count++;
}

// ================================
// DEQUEUE — Circular Queue
// ================================
infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
```

```
    }

    infotype x = Q.info[Q.head];

    if (Q.count == 1) {
        Q.head = Q.tail = -1;
    }
    else {
        Q.head = (Q.head + 1) % MAX;
    }

    Q.count--;
    return x;
}

// ================================
// PRINT — Tetap sama outputnya dengan Alternatif 1
// ================================
void printInfo(Queue Q) {
    cout << Q.head << "   " << Q.tail << "   ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    // mencetak sesuai urutan queue (head → tail)
    int i = Q.head;
    for (int c = 0; c < Q.count; c++) {
        cout << Q.info[i] << " ";
        i = (i + 1) % MAX;
    }
    cout << endl;
}
```

Screenshots Output

```
PS C:\Users\ASUS\Videos\strukdat\Alternatif 3> g++ main.cpp queue.cpp -o main
>> .\main
>>
Hello World!
---------------------
H - T    | Queue info
---------------------
-1   -1   empty queue
0   0   5
0   1   5 2
0   2   5 2 7
1   2   2 7
1   3   2 7 4
2   3   7 4
3   3   4
PS C:\Users\ASUS\Videos\strukdat\Alternatif 3>
```

Deskripsi:

Kode tersebut mengimplementasikan struktur data circular queue (antrian melingkar) berbasis array statis dengan ukuran maksimum MAX = 5 yang memungkinkan pemanfaatan kembali indeks yang sudah kosong melalui operasi modulo. Struktur Queue terdiri dari array info untuk menyimpan data, dua penunjuk indeks yaitu head (elemen terdepan) dan tail (elemen terakhir), serta variabel count untuk melacak jumlah elemen saat ini sehingga pengecekan kosong (isEmptyQueue) dan penuh (isFullQueue) menjadi lebih akurat. Fungsi createQueue menginisialisasi queue dalam keadaan kosong dengan head dan tail bernilai -1 dan count bernilai 0. Proses enqueue menambahkan elemen baru di posisi tail, di mana jika queue kosong maka head dan tail diset ke 0, sedangkan jika tidak kosong maka tail digeser ke kanan secara melingkar menggunakan rumus (tail + 1) % MAX, lalu count ditambah. Proses dequeue mengambil elemen di posisi head dan menggeser head ke posisi berikutnya secara melingkar dengan (head + 1) % MAX, atau mengosongkan queue kembali (mengatur head dan tail ke -1) jika hanya tersisa satu elemen, kemudian count dikurangi. Fungsi printInfo menampilkan posisi head dan tail serta seluruh isi queue sesuai urutan antrian dari depan ke belakang dengan bantuan perulangan melingkar berdasarkan count. Pada fungsi main, operasi enqueue dan dequeue dijalankan secara berurutan untuk menunjukkan perubahan isi queue dan pergerakan indeks secara dinamis, sehingga program ini menjadi contoh lengkap penerapan circular queue tanpa perlu melakukan pergeseran elemen di dalam array.

E. Kesimpulan

Secara keseluruhan, semua kode di atas menunjukkan berbagai cara implementasi struktur data queue (antrian) menggunakan array yang memiliki prinsip dasar FIFO (First In First Out), di mana elemen yang pertama masuk akan menjadi elemen yang pertama keluar. Perbedaannya terletak pada cara pengelolaan indeks head dan tail: ada versi linear sederhana yang hanya menggeser head ke depan saat dequeue tanpa melakukan shifting data, ada versi di mana tail bergerak saat enqueue sementara head bergerak saat dequeue, dan versi paling efektif yaitu circular queue yang memanfaatkan operasi modulo (%) untuk memutar kembali head dan tail ke awal array ketika sudah mencapai batas akhir,

sehingga ruang kosong dapat digunakan kembali secara optimal. Semua versi menggunakan fungsi utama seperti createQueue, enqueue, dequeue, isEmpty, isFull, dan printInfo untuk mengelola dan menampilkan isi antrian. Dengan demikian, rangkaian kode tersebut tidak hanya memperlihatkan prinsip kerja queue, tetapi juga memperbandingkan efektivitas antara queue linear biasa dan circular queue dalam pemanfaatan memori serta efisiensi pergerakan indeks.

F. Referensi

Queue (abstract data type)

https://en.wikipedia.org/wiki/Queue_%28abstract_data_type%29

Introduction to Circular Queue. *https://www.geeksforgeeks.org/dsa/introduction-to-circular-queue-1/*