LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :
NAMA : Muhammad Omar Nadiv
NIM : 103112430063

Dosen
Fahrudin Mukti Wibowo, S.Kom., M.Eng.

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue atau antrian merupakan salah satu struktur data linier yang menerapkan prinsip First In, First Out (FIFO), yaitu elemen yang pertama kali masuk akan menjadi elemen pertama yang keluar. Struktur ini bekerja seperti antrian pada kehidupan nyata, di mana setiap elemen hanya dapat ditambahkan dari bagian belakang (rear/tail) melalui operasi enqueue, dan hanya dapat dihapus dari bagian depan (front/head) melalui operasi dequeue.

Queue dapat diimplementasikan menggunakan array, linked list, maupun circular array, tergantung kebutuhan dan efisiensi memori. Struktur ini banyak digunakan dalam berbagai aplikasi, seperti sistem antrian proses pada sistem operasi, buffer data pada jaringan komputer, manajemen tugas pada printer, serta simulasi sistem pelayanan. Operasi pada queue secara umum memiliki kompleksitas O(1), karena proses penambahan dan penghapusan hanya melibatkan ujung-ujung tertentu tanpa perlu memindahkan seluruh elemen. Dengan sifatnya yang teratur dan efisien, queue menjadi salah satu struktur data fundamental yang sering digunakan dalam pemrograman dan algoritma.

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

Guided 1 (file queue.h)

```
#ifndef QUEUE_H
#define QUEUE_H

#define MAX_QUEUE 5

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

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
```

Guided 1 (file queue.cpp)

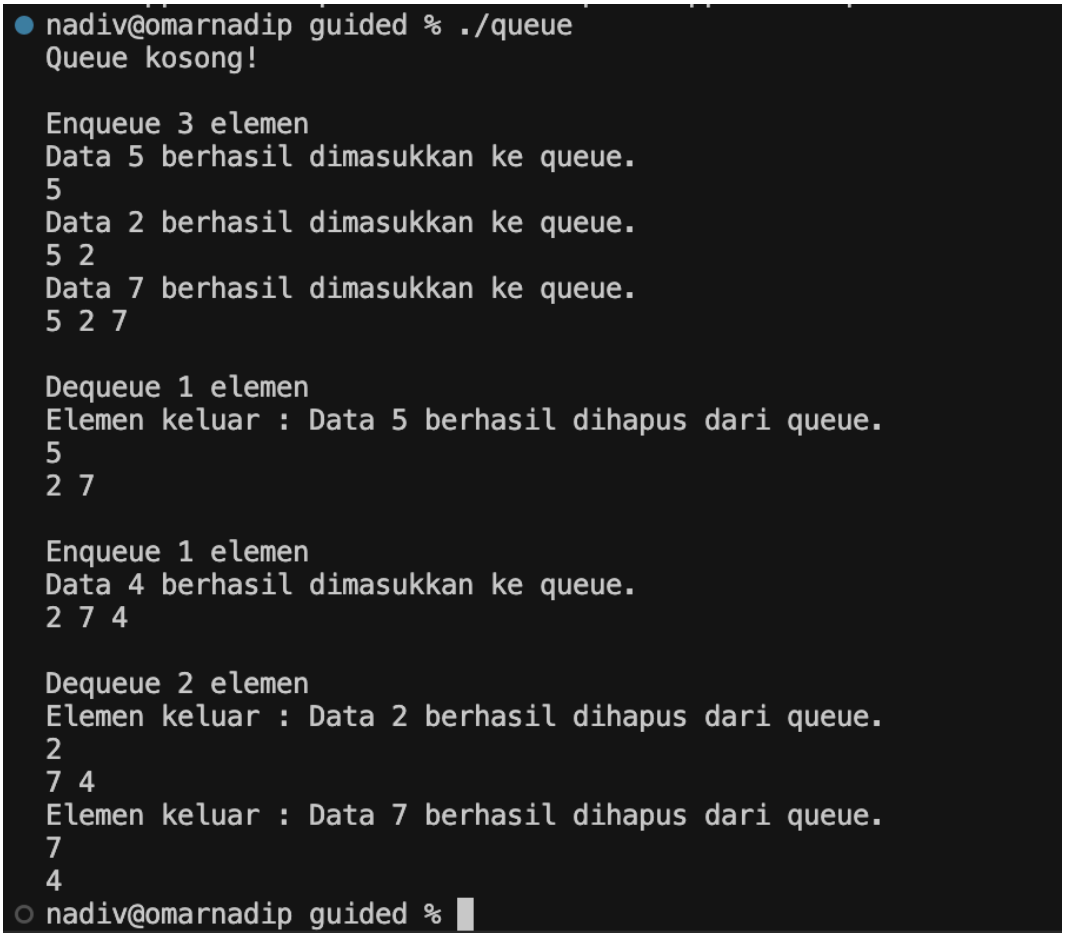
```
#include "queue.h"
#include <iostream>
using namespace std;

void createQueue(Queue &Q)
{
   Q.head = 0;
   Q.tail = 0;
   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))
   {
       Q.info[Q.tail] = x;
       Q.tail = (Q.tail + 1) % MAX_QUEUE;
       Q.count++;
       cout << "Data " << x << " berhasil dimasukkan ke queue." << endl;
   }
   else
   {
```

```
        cout << "Queue penuh!" << endl;
    }
}

int dequeue(Queue &Q)
{
    if (!isEmpty(Q))
    {
        int x = Q.info[Q.head];
        Q.head = (Q.head + 1) % MAX_QUEUE;
        Q.count--;
        cout << "Data " << x << " berhasil dihapus dari queue." << endl;
        return x;
    }
    else
    {
        cout << "Queue kosong!" << endl;
        return -1;
    }
}

void printInfo(Queue Q)
{
    if (!isEmpty(Q))
    {
        int i = Q.head;
        int c = 0;
        while (c < Q.count)
        {
            cout << Q.info[i] << " ";
            i = (i + 1) % MAX_QUEUE;
            c++;
        }
        cout << endl;
    }
    else
    {
        cout << "Queue kosong!" << endl;
    }
}
```

Guided 1 (file main.cpp)

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

int main()
{
   Queue Q;

   createQueue(Q);
   printInfo(Q);

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

   cout << "\nDequeue 1 elemen" << endl;
   cout << "Elemen keluar : " << dequeue(Q) << endl;
   printInfo(Q);

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

   cout << "\nDequeue 2 elemen" << endl;
   cout << "Elemen keluar : " << dequeue(Q) << endl;
   printInfo(Q);
   cout << "Elemen keluar : " << dequeue(Q) << endl;
   printInfo(Q);

   return 0;
}
```

Screenshots Output



Deskripsi:

Program ini digunakan untuk mengelola data dalam struktur data Queue menggunakan bahasa C++. Queue diimplementasikan dengan menggunakan array berukuran tetap dan menggunakan pendekatan circular queue agar pemanfaatan indeks lebih efisien. Struktur Queue menyimpan elemen dalam array info, serta memiliki tiga variabel penting yaitu head, tail, dan count yang digunakan untuk melacak posisi elemen depan, elemen belakang, dan jumlah elemen dalam antrian. Fungsi enqueue digunakan untuk menambahkan elemen ke posisi tail dengan pergerakan indeks secara melingkar menggunakan operasi modulo, sedangkan dequeue menghapus elemen dari posisi head. Fungsi isEmpty dan isFull digunakan untuk memeriksa kondisi queue, dan fungsi printInfo menampilkan elemen-elemen queue sesuai urutan FIFO. Program ini memperlihatkan cara kerja queue secara lengkap mulai dari penambahan, penghapusan, hingga penelusuran data dalam antrian.

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

Alternatif 1 (file queue.h)

```
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

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

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
bool isFullQueue(Queue Q);
void enqueue(Queue &Q, int x);
int dequeue(Queue &Q);
void printInfo(Queue Q);

#endif
```

Alternatif 1 (file queue.cpp)

```
#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;
}
```

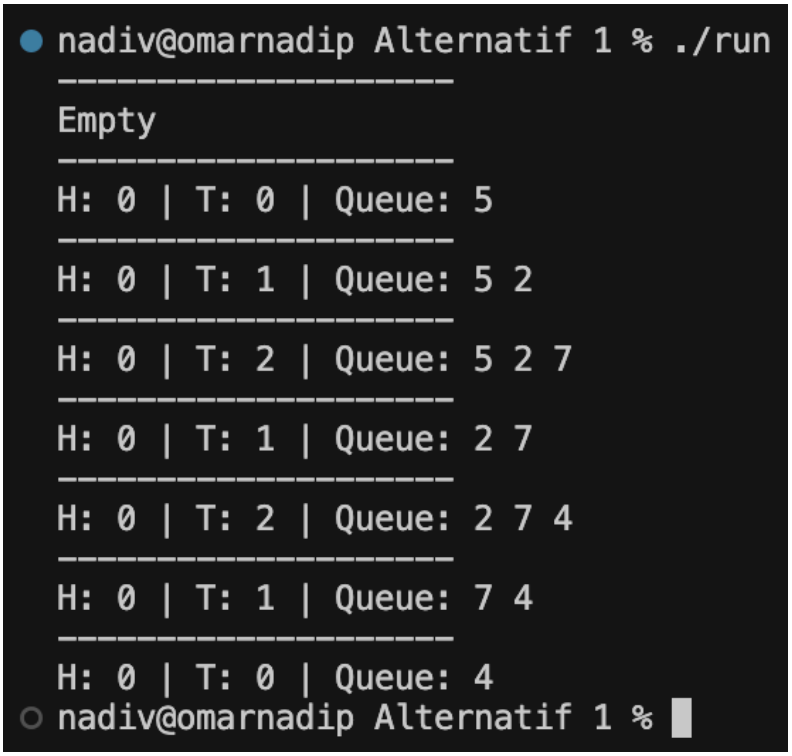
```
bool isFullQueue(Queue Q) {
   return Q.tail == MAX - 1;
}

void enqueue(Queue &Q, int x) {
   if (isFullQueue(Q)) {
       cout << "Queue penuh!\n";
       return;
   }

   if (isEmptyQueue(Q)) {
       Q.head = 0;
       Q.tail = 0;
       Q.info[0] = x;
   } else {
       Q.tail++;
       Q.info[Q.tail] = x;
   }
}

int dequeue(Queue &Q) {
   if (isEmptyQueue(Q)) {
       cout << "Queue kosong!\n";
       return -1;
   }

   int x = Q.info[0];

   for (int i = 0; i < Q.tail; i++) {
       Q.info[i] = Q.info[i + 1];
   }

   Q.tail--;

   if (Q.tail < 0) {
       Q.head = Q.tail = -1;
   }

   return x;
}
```

```
void printInfo(Queue Q) {
   cout << "--------------------\n";
   if (isEmptyQueue(Q)) {
       cout << "Empty\n";
       return;
   }

   cout << "H: " << Q.head << " | T: " << Q.tail << " | Queue: ";
   for (int i = 0; i <= Q.tail; i++) {
       cout << Q.info[i] << " ";
   }
   cout << endl;
}
```

Alternatif 1 (file main.cpp)

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

int main() {
   Queue Q;
   CreateQueue(Q);

   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;
}
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data Queue menggunakan metode dasar di mana indeks head tetap berada pada posisi 0 dan operasi dequeue dilakukan dengan cara menggeser seluruh elemen ke kiri. Struktur Queue terdiri dari array info, serta indeks head dan tail yang digunakan untuk menentukan posisi elemen dalam antrian. Fungsi CreateQueue menginisialisasi queue agar kosong, isEmptyQueue dan isFullQueue mengecek kondisi queue, enqueue menambahkan elemen ke bagian belakang selama belum penuh, dan dequeue menghapus elemen pertama sekaligus melakukan shifting untuk menjaga head tetap pada indeks 0. Fungsi printInfo menampilkan posisi head, tail, dan seluruh elemen queue. Implementasi Alternatif 1 ini merupakan bentuk queue paling sederhana, namun kurang efisien karena membutuhkan pergeseran elemen setiap kali melakukan operasi dequeue.

Alternatif 2 (file queue.h)

```
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

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

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
bool isFullQueue(Queue Q);
void compress(Queue &Q);
void enqueue(Queue &Q, int x);
int dequeue(Queue &Q);
void printInfo(Queue Q);

#endif
```

Alternatif 2 (file queue.cpp)

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

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

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

bool isFullQueue(Queue Q) {
   return (Q.head == 0 && Q.tail == MAX - 1);
```

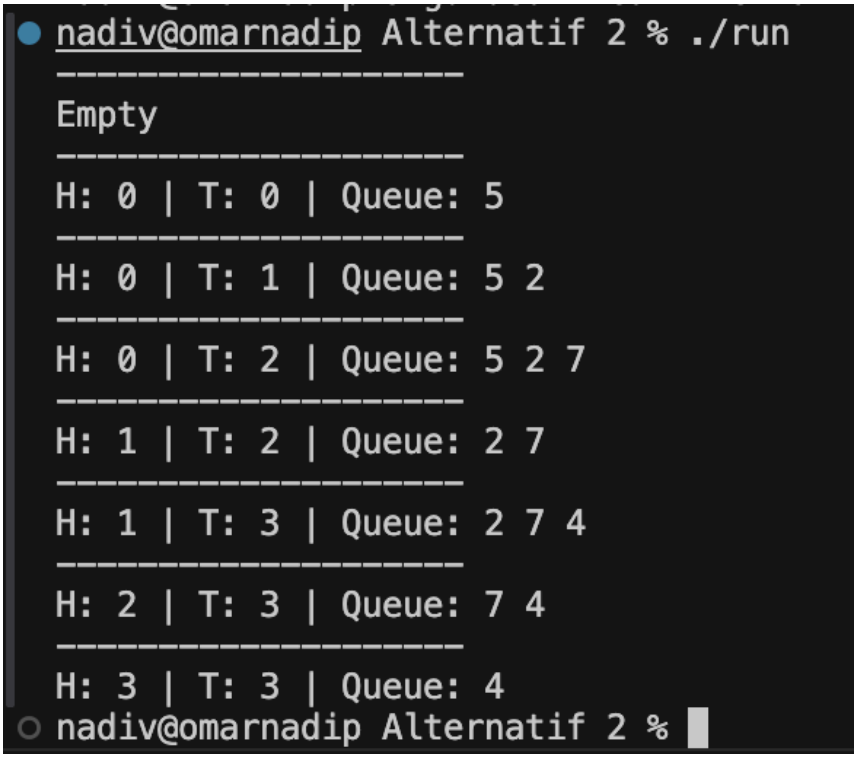
```
}

void compress(Queue &Q) {
   int j = 0;
   for (int i = Q.head; i <= Q.tail; i++) {
       Q.info[j++] = Q.info[i];
   }
   Q.head = 0;
   Q.tail = j - 1;
}

void enqueue(Queue &Q, int x) {
   if (isFullQueue(Q)) {
       cout << "Queue penuh!\n";
       return;
   }

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

       Q.tail++;
   }

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

int dequeue(Queue &Q) {
   if (isEmptyQueue(Q)) {
       cout << "Queue kosong!\n";
       return -1;
   }

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

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

```
    }

    return x;
}

void printInfo(Queue Q) {
    cout << "--------------------\n";

    if (isEmptyQueue(Q)) {
        cout << "Empty\n";
        return;
    }

    cout << "H: " << Q.head << " | T: " << Q.tail << " | Queue: ";
    for (int i = Q.head; i <= Q.tail; i++)
        cout << Q.info[i] << " ";
    cout << endl;
}
```

Alternatif 2 (file main.cpp)

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

int main() {
    Queue Q;
    CreateQueue(Q);

    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;
}
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data Queue menggunakan pendekatan di mana indeks head dan tail dapat bergerak maju, namun tetap menggunakan array biasa tanpa circular. Ketika terjadi operasi dequeue, head cukup digeser ke indeks berikutnya tanpa melakukan shifting elemen, sehingga lebih efisien dibanding Alternatif 1. Namun, apabila tail sudah berada di posisi terakhir array sementara masih ada ruang di bagian depan (karena beberapa elemen sudah didequeue), program menggunakan fungsi compress untuk menggeser seluruh elemen agar kembali dimulai dari indeks 0 sehingga ruang array dapat digunakan kembali. Fungsi CreateQueue menginisialisasi queue, isEmptyQueue dan isFullQueue memeriksa kondisi antrian, enqueue menambahkan elemen baru, dequeue menghapus elemen depan, dan printInfo menampilkan isi queue. Alternatif 2 ini menawarkan penggunaan memori yang lebih optimal dibanding Alternatif 1, karena shifting hanya dilakukan ketika benar benar diperlukan.

Alternatif 3 (file queue.h)

```
#ifndef QUEUE_H
#define QUEUE_H

const int MAX = 5;

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

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
bool isFullQueue(Queue Q);
void enqueue(Queue &Q, int x);
int dequeue(Queue &Q);
void printInfo(Queue Q);

#endif
```

Alternatif 3 (file queue.cpp)

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

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

   for (int i = 0; i < MAX; i++) {
       Q.info[i] = 0;
   }
}

bool isEmptyQueue(Queue Q) {
   return Q.head == -1;
```

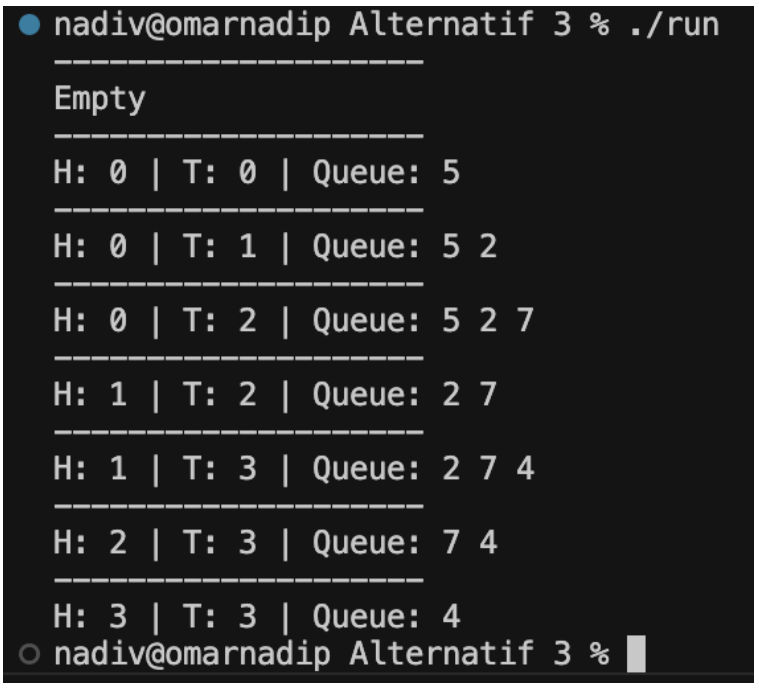
```
}

bool isFullQueue(Queue Q) {
   return ((Q.tail + 1) % MAX) == Q.head;
}

void enqueue(Queue &Q, int x) {
   if (isFullQueue(Q)) {
       cout << "Queue penuh!\n";
       return;
   }

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

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

int dequeue(Queue &Q) {
   if (isEmptyQueue(Q)) {
       cout << "Queue kosong!\n";
       return -1;
   }

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

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

   return x;
}

void printInfo(Queue Q) {
   cout << "--------------------\n";
```

```
    if (isEmptyQueue(Q)) {
        cout << "Empty\n";
        return;
    }

    cout << "H: " << Q.head << " | T: " << Q.tail << " | Queue: ";

    int i = Q.head;
    while (true) {
        cout << Q.info[i] << " ";
        if (i == Q.tail) break;
        i = (i + 1) % MAX;
    }

    cout << endl;
}
```

Alternatif 3 (file main.cpp)

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

int main() {
    Queue Q;
    CreateQueue(Q);

    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;
}
```

Screenshots Output



Deskripsi:

Program ini mengimplementasikan struktur data Queue menggunakan metode circular queue, di mana indeks head dan tail bergerak secara melingkar (modulo) di dalam array sehingga ruang penyimpanan dapat dimanfaatkan lebih efisien tanpa perlu melakukan shifting atau kompresi seperti pada alternatif sebelumnya. Ketika queue kosong, head dan tail diinisialisasi ke posisi 0, dan setiap operasi enqueue akan menambahkan elemen ke indeks tail yang diperbarui menggunakan (tail + 1) % MAX. Operasi dequeue menghapus elemen dari indeks head yang juga bergerak menggunakan modulo, dan ketika elemen terakhir dihapus, queue kembali ke keadaan kosong. Fungsi isFullQueue memeriksa kondisi penuh dengan membandingkan posisi tail berikutnya dengan head, sedangkan isEmptyQueue mengecek apakah queue tidak memiliki

elemen. Fungsi printInfo menampilkan elemen queue sesuai urutan dengan memperhitungkan pergerakan circular. Alternatif 3 ini merupakan implementasi queue yang paling efisien karena memanfaatkan seluruh kapasitas array tanpa perlu memindahkan elemen secara manual.

D. Kesimpulan

Pada modul 8 ini, saya mempelajari penggunaan struktur data Queue dalam pemrograman C++. Saya memahami konsep dasar queue yang menerapkan prinsip First In, First Out (FIFO), di mana elemen yang pertama masuk akan menjadi elemen pertama yang keluar. Selain itu, saya belajar mengimplementasikan queue menggunakan array melalui tiga pendekatan berbeda, yaitu queue sederhana dengan shifting, queue dengan kompresi, serta circular queue yang lebih efisien. Saya juga mempraktikkan operasi dasar seperti enqueue, dequeue, pengecekan kondisi kosong dan penuh, serta menampilkan isi queue menggunakan printInfo. Melalui modul ini, saya memperoleh pemahaman yang lebih baik mengenai cara kerja antrian dalam pengolahan data dan bagaimana struktur ini digunakan dalam berbagai aplikasi yang membutuhkan pengelolaan data secara terurut.

E. Referensi

https://en.wikipedia.org/wiki/Queue_(abstract_data_type)

https://www.kompas.com/tekno/read/2022/12/01/02150047/pengertian-stack-dan-queue-serta-contoh-penerapannya

https://www.mahirkoding.com/struktur-data-queue-dan-implementasinya