LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :
Nama: Dina Nadhyfa
NIM : 103112430052

Dosen
Fahrudin Mukti Wibowo

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Queue adalah struktur data linear yang mengikuti prinsip FIFO (First In, First Out), di mana elemen yang pertama kali dimasukkan akan menjadi elemen pertama yang dikeluarkan. Dalam queue, elemen baru selalu ditambahkan di bagian belakang (rear/back) dan elemen dihapus dari bagian depan (front). Struktur ini banyak digunakan untuk menangani antrian data dalam berbagai aplikasi, seperti buffer perangkat keras untuk mengelola perbedaan kecepatan antara CPU dan perangkat input/output. Implementasi queue dapat dilakukan dengan menggunakan array atau linked list, namun dalam bahasa pemrograman C++ telah tersedia implementasi queue yang efisien melalui Standard Template Library (STL) menggunakan kelas std::queue<T>, di mana T adalah tipe elemen yang disimpan.

Operasi dasar pada queue di C++ meliputi penambahan elemen dengan fungsi push() (enqueue), penghapusan elemen dengan fungsi pop() (dequeue), serta akses ke elemen depan dan belakang dengan fungsi front() dan back(). Selain itu, terdapat fungsi untuk memeriksa apakah queue kosong (empty()) dan ukuran queue (size()), di mana semua operasi ini memiliki kompleksitas waktu O(1). Queue sering digunakan dalam berbagai algoritma dan aplikasi, seperti pemrograman level-order traversal pohon, Breadth-First Search (BFS) pada graf, penjadwalan CPU, dan manajemen memori. Dalam konteks praktikum C++, queue sangat berguna untuk mensimulasikan antrian proses atau buffer data secara real-time. Semua informasi ini mendukung pemahaman dan implementasi praktis queue dalam pengembangan program berbasis C++.

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

Queue.h

```
#ifndef QUEUE_H // Jika QUEUE_H belum didefinisikan
#define QUEUE_H // maka definisikan QUEUE_H untuk mencegah multiple
inclusion

#define MAX_QUEUE 5 // menentukan ukuran maksimum queue

//mendefinisikan tipe data Queue
struct Queue {
    int info[MAX_QUEUE];
    int head;
    int tail;
    int count;
};

void createQueue(Queue &Q); // prosedur untuk membuat queue kosong
bool isEmpty(Queue Q); // fungsi untuk mengecek apakah queue kosong
bool isFull(Queue Q); // fungsi untuk mengecek apakah queue penuh
void enqueue(Queue &Q, int x); // prosedur untuk menambahkan elemen ke
dalam queue
```

```
int dequeue(Queue &Q); // fungsi untuk menghapus elemen dari queue
void printInfo(Queue Q); // prosedur untuk menampilkan isi queue

#endif // akhir dari definisi QUEUE_H
```

Queue.cpp

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

using namespace std; // menggunakanan namespace std agar tidak perlu
menulis std:: setiap kali

// definisi prosedur dan fungsi 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 0
}

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

// definisi prosedur untuk menambahkan elemen ke dalam queue (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 memutar
        Q.tail = (Q.tail + 1) % MAX_QUEUE; // update posisi ekor secara
melingkar
        Q.count++; // tambah jumlah elemen
    } else { //jika queue penuh
        cout << "Antrean penuh!" << endl; // tampilkan pesan jika queue
penuh
    }
}

// definisi fungsi untuk menghapus elemen dari queue (dequeue)
int dequeue(Queue &Q) {
```

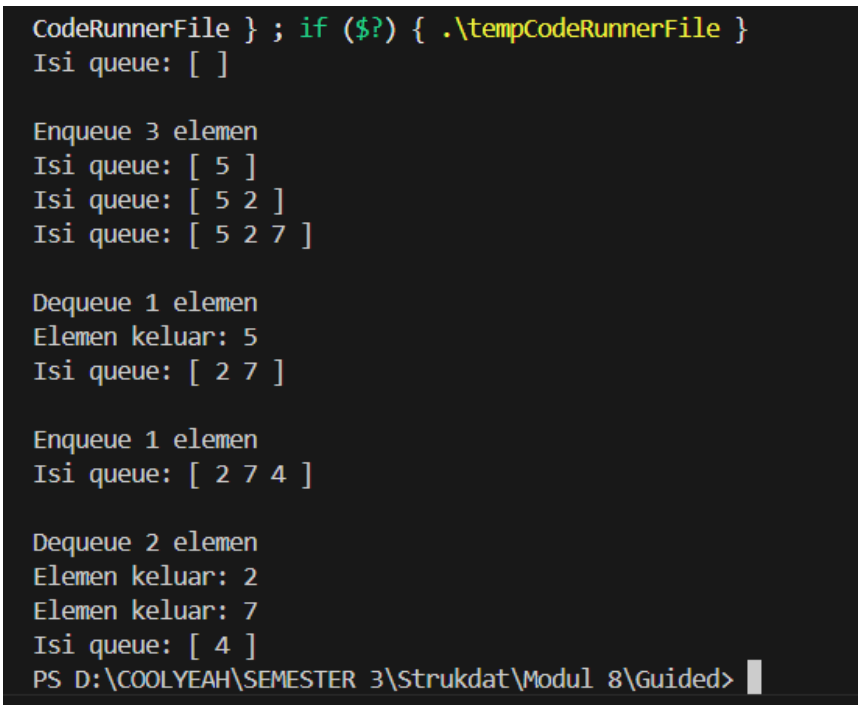
```
    if (!isEmpty(Q)) { // jika queue tidak kosong
        int x = Q.info[Q.head]; // ambil data dari posisi kepala (head)
        // pindahkan kepala secara memutar
        Q.head = (Q.head + 1) % MAX_QUEUE; // update posisi kepala
secara melingkar
        Q.count--; // kurangi jumlah elemen
        return x; // kembalikan data yang dihapus
    } else { // jika queue kosong
        cout << "Antrean kosong!" << endl; // tampilkan pesan jika queue
kosong
        return -1; // kembalikan nilai -1 sebagai indikasi queue kosong
    }
}

// definisi prosedur untuk menampilkan isi queue
void printInfo(Queue Q) {
    cout << "Isi queue: [ "; // tampilkan judul
    if (!isEmpty(Q)) {
        int i = Q.head;
        int n = 0;
        while (n < Q.count) {
            cout << Q.info[i] << " ";
            i = (i + 1) % MAX_QUEUE;
            n++;
        }
    }
    cout << "]" << endl; // akhiri tampilan isi queue
}
```

Main.cpp

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

using namespace std; // menggunakanan namespace std agar tidak perlu
menulis std:: setiap kali

// fungsi utama program
int main() {
    Queue Q; // deklarasi variabel Q bertipe Queue

    createQueue(Q); // membuat queue kosong
    printInfo(Q); // menampilkan isi queue (harusnya kosong)

    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;
    // hapus 1 elemen dari queue dan menampilkannya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q); // menampilkan isi queue setelah dequeue

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

    cout << "\nDequeue 2 elemen" << endl;
    // hapus 1 elemen dan tampilkannya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    // hapus 1 elemen dan tampilkannya
    cout << "Elemen keluar: " << dequeue(Q) << endl;
    printInfo(Q); // menampilkan isi queue setelah dequeue

    return 0; // mengembalikan nilai 0 menandakan program selesai dengan
sukses
}
```

Screenshots Output



Deskripsi:

Program di atas merupakan implementasi struktur data Queue (antrian) menggunakan array statis dengan konsep circular queue dalam bahasa C++. Queue direpresentasikan oleh struct Queue yang memiliki array info untuk menyimpan elemen, serta variabel head, tail, dan count untuk mengatur posisi elemen dan jumlah data dalam antrian. Program menyediakan operasi dasar queue, yaitu createQueue untuk inisialisasi queue kosong, isEmpty dan isFull untuk mengecek kondisi queue, enqueue untuk menambahkan elemen ke bagian belakang antrian, dan dequeue untuk menghapus elemen dari bagian depan antrian sesuai prinsip FIFO (First In First Out). Mekanisme circular queue diterapkan dengan operasi modulo (% MAX_QUEUE) agar indeks array dapat berputar kembali ke awal saat mencapai batas maksimum. Selain itu, prosedur printInfo digunakan untuk menampilkan isi queue dari posisi head hingga jumlah elemen saat ini. Pada fungsi main, program mendemonstrasikan proses enqueue dan dequeue secara bertahap, dimulai dari queue kosong, penambahan beberapa elemen, penghapusan elemen, hingga penambahan kembali elemen, sehingga memperlihatkan bahwa queue bekerja dengan benar sesuai konsep FIFO dan circular array.

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

Unguided 1

Unguided-queue.h

```h
#ifndef QUEUE_H_INCLUDED
#define QUEUE_H_INCLUDED
#include <iostream>

using namespace std;

typedef int infotype;

struct Queue {
    infotype info[5];
    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
```

Unguided-queue-1.cpp

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

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

// Mengecek apakah queue kosong
bool isEmptyQueue(Queue Q) {
    return (Q.head == -1 && Q.tail == -1);
}

// Mengecek apakah queue penuh
bool isFullQueue(Queue Q) {
    return (Q.tail == 4); // Index maksimal array adalah 4 (0-4)
}

// Menambahkan elemen ke queue
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            Q.head = 0;
            Q.tail = 0;
        } else {
            Q.tail++;
        }
        Q.info[Q.tail] = x;
    }
}

// Menghapus elemen dari queue
infotype dequeue(Queue &Q) {
    infotype x = 0;
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
    } else {
        x = Q.info[Q.head];

        // ALTERNATIF 1: head diam, tail bergerak (dengan pergeseran)
        if (Q.head == Q.tail) {
            // Queue hanya memiliki 1 elemen
            Q.head = -1;
```

```
            Q.tail = -1;
        } else {
            // Geser semua elemen ke depan
            for (int i = Q.head; i < Q.tail; i++) {
                Q.info[i] = Q.info[i + 1];
            }
            Q.tail--;
        }
    }
    return x;
}

// Menampilkan isi queue
void printInfo(Queue Q) {
    cout << endl;
    cout << " " << Q.head << " - " << Q.tail << " | ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue";
    } else {
        // ALTERNATIF 1: tampilkan dari head sampai tail
        for (int i = Q.head; i <= Q.tail; i++) {
            cout << Q.info[i];
            if (i < Q.tail) cout << " ";
        }
    }
}
```

Main.cpp

```
#include <iostream>
#include "unguided-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);
    dequeue(Q); printInfo(Q);
    enqueue(Q, 4); printInfo(Q);
```

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

    return 0;
}
```

Screenshoot Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 8\Unguided> g++ main.cpp unguided-queue.cpp -o main
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 8\Unguided> .\main
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 - 0 | 7
 0 - 1 | 7 4
 0 - 0 | 4
 -1 - -1 | empty queue
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 8\Unguided>
```

Deskripsi:

Program di atas merupakan implementasi struktur data queue menggunakan array statis, di mana antrian dikelola dengan dua indeks yaitu head dan tail yang diawali dengan nilai -1 sebagai penanda queue kosong. Pada metode ini, proses enqueue dilakukan dengan menambahkan elemen pada posisi tail yang terus bergerak maju, sedangkan head tetap berada di posisi awal. Ketika terjadi operasi dequeue, elemen pada posisi head dikeluarkan sesuai prinsip FIFO, kemudian seluruh elemen yang tersisa digeser satu posisi ke depan agar elemen berikutnya tetap berada di indeks head. Program juga menyediakan fungsi pengecekan kondisi queue kosong dan penuh, serta prosedur untuk menampilkan isi queue beserta posisi head dan tail. Hasil pengujian menunjukkan bahwa queue bekerja dengan benar sesuai konsep FIFO, namun metode head diam ini kurang efisien karena membutuhkan proses pergeseran elemen setiap kali dilakukan dequeue.

Unguided 2

Header dan main program tidak ada yang di ubah.

Unguided-queue-2.cpp

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

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

// Mengecek apakah queue kosong
bool isEmptyQueue(Queue Q) {
    return (Q.head == -1 && Q.tail == -1);
}

// Mengecek apakah queue penuh (Alternatif 2)
bool isFullQueue(Queue Q) {
    // Queue penuh jika tail sudah mencapai index maksimal (4)
    return (Q.tail == 4);
}

// Menambahkan elemen ke queue (Alternatif 2)
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            // Queue kosong, inisialisasi head dan tail
            Q.head = 0;
            Q.tail = 0;
        } else {
            // Tail bergerak maju
            Q.tail++;
        }
        Q.info[Q.tail] = x;
    }
}

// Menghapus elemen dari queue (Alternatif 2)
infotype dequeue(Queue &Q) {
    infotype x = 0;
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
    } else {
```

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

        // ALTERNATIF 2: head bergerak, tail diam (tanpa pergeseran)
        if (Q.head == Q.tail) {
            // Queue hanya memiliki 1 elemen
            Q.head = -1;
            Q.tail = -1;
        } else {
            // Head bergerak maju tanpa menggeser elemen
            Q.head++;
        }
    }
    return x;
}

// Menampilkan isi queue (Alternatif 2)
void printInfo(Queue Q) {
    cout << " " << Q.head << " - " << Q.tail << " | ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue";
    } else {
        // ALTERNATIF 2: tampilkan dari head sampai tail secara linear
        for (int i = Q.head; i <= Q.tail; i++) {
            cout << Q.info[i];
            if (i < Q.tail) cout << " ";
        }
    }
    cout << endl;
}
```

Screenshot Output

```
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 8\Unguided> .\main.exe
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
 2 - 2 | 7
 2 - 3 | 7 4
 3 - 3 | 4
 -1 - -1 | empty queue
PS D:\COOLYEAH\SEMESTER 3\Strukdat\Modul 8\Unguided>
```

Deskripsi:

Program Alternatif 2 menggunakan pendekatan head bergerak, tail bergerak tanpa pergeseran elemen. Ketika elemen ditambahkan dengan enqueue(), tail bergerak maju secara linear (tail++) dan elemen disimpan di posisi tail. Ketika elemen dihapus dengan dequeue(), nilai di posisi head diambil kemudian head bergerak maju (head++) tanpa menggeser elemen-elemen lainnya, sehingga operasi dequeue sangat efisien dengan kompleksitas O(1). Kelemahan metode ini adalah adanya "lubang" atau ruang kosong di awal array yang tidak dapat digunakan lagi, sehingga queue bisa menjadi penuh (ketika tail == 4) meskipun masih ada slot kosong di index awal array, yang membuat penggunaan memori kurang optimal dibanding Alternatif 1 yang menggunakan pergeseran atau Alternatif 3 yang menggunakan circular queue.

Unguided 3

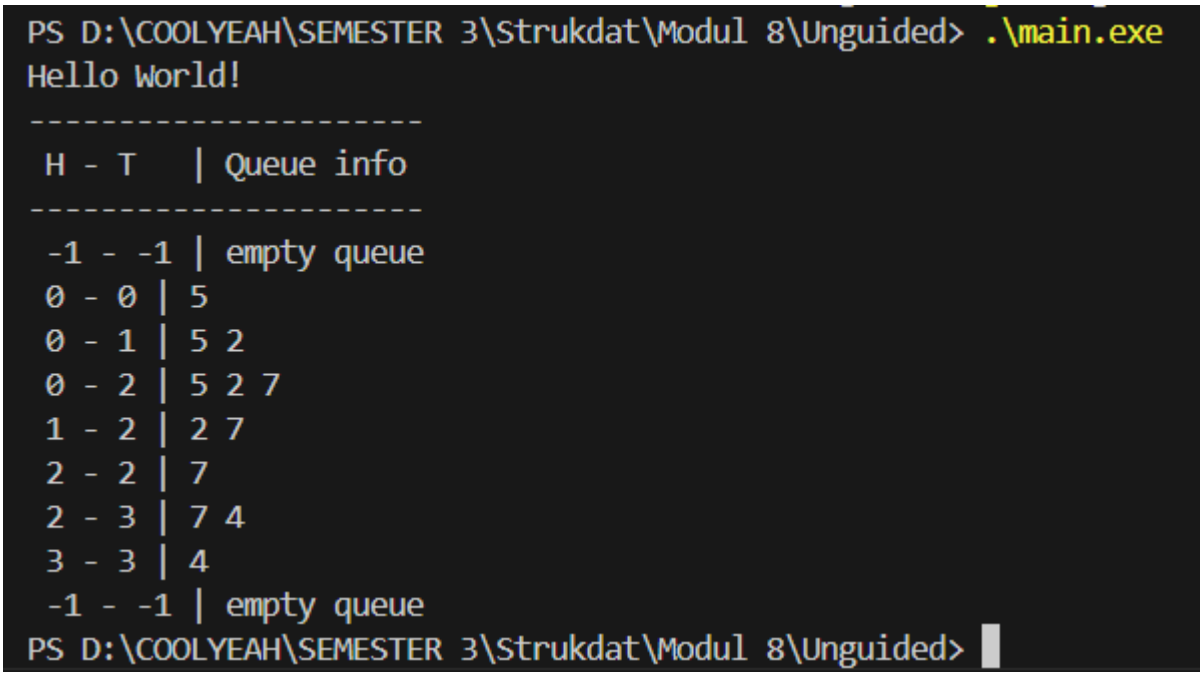
Header dan main program tifdak diubah.

Unguided-queue-3.cpp

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

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

// Mengecek apakah queue kosong
bool isEmptyQueue(Queue Q) {
    return (Q.head == -1 && Q.tail == -1);
}

// Mengecek apakah queue penuh (Alternatif 3 - Circular)
bool isFullQueue(Queue Q) {
    // Queue penuh jika posisi berikutnya dari tail adalah head
    return ((Q.tail + 1) % 5 == Q.head);
}

// Menambahkan elemen ke queue (Alternatif 3 - Circular)
void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh!" << endl;
    } else {
        if (isEmptyQueue(Q)) {
            // Queue kosong, inisialisasi head dan tail
            Q.head = 0;
            Q.tail = 0;
```

```
        } else {
            // Tail berputar (circular) menggunakan modulo
            Q.tail = (Q.tail + 1) % 5;
        }
        Q.info[Q.tail] = x;
    }
}

// Menghapus elemen dari queue (Alternatif 3 - Circular)
infotype dequeue(Queue &Q) {
    infotype x = 0;
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong!" << endl;
        return -1;
    } else {
        x = Q.info[Q.head];

        // ALTERNATIF 3: head dan tail berputar (circular)
        if (Q.head == Q.tail) {
            // Queue hanya memiliki 1 elemen
            Q.head = -1;
            Q.tail = -1;
        } else {
            // Head berputar (circular) menggunakan modulo
            Q.head = (Q.head + 1) % 5;
        }
    }
    return x;
}

// Menampilkan isi queue (Alternatif 3 - Circular)
void printInfo(Queue Q) {
    cout << " " << Q.head << " - " << Q.tail << " | ";

    if (isEmptyQueue(Q)) {
        cout << "empty queue";
    } else {
        // ALTERNATIF 3: tampilkan dengan cara berputar (circular)
        int i = Q.head;
        while (true) {
            cout << Q.info[i];
            if (i == Q.tail) break;
            cout << " ";
            i = (i + 1) % 5; // Berputar menggunakan modulo
        }
    }
    cout << endl;
}
```

Screenshot Output



Deskripsi:

Program di atas menggunakan pendekatan head dan tail berputar dengan memanfaatkan operasi modulo untuk membuat array seolah-olah berbentuk lingkaran. Ketika elemen ditambahkan dengan enqueue(), tail bergerak maju menggunakan formula tail = (tail + 1) % 5 sehingga setelah mencapai index 4, tail akan kembali ke index 0 jika ada ruang kosong. Begitu pula dengan dequeue(), head bergerak maju menggunakan head = (head + 1) % 5 sehingga berputar kembali ke awal array. Queue dianggap penuh ketika (tail + 1) % 5 == head, yang berarti posisi berikutnya dari tail akan bertabrakan dengan head. Keunggulan metode ini adalah efisiensi waktu O(1) untuk kedua operasi enqueue dan dequeue tanpa perlu pergeseran elemen, sekaligus penggunaan memori yang optimal karena seluruh slot array dapat digunakan kembali secara berputar, menjadikannya implementasi queue yang paling efisien dibanding Alternatif 1 dan 2.

D. Kesimpulan

Dalam implementasi struktur data Queue menggunakan array, terdapat tiga alternatif pendekatan yang masing-masing memiliki karakteristik dan trade-off berbeda. Alternatif 1 (head diam, tail bergerak dengan pergeseran) mempertahankan head selalu di index 0 dengan cara menggeser semua elemen ke kiri setiap kali terjadi dequeue, sehingga operasi dequeue memiliki kompleksitas O(n) karena perlu pergeseran, namun penggunaan memori lebih optimal karena selalu dimulai dari index 0 tanpa ada ruang kosong yang terbuang. Alternatif 2 (head bergerak, tail bergerak tanpa pergeseran) meningkatkan efisiensi operasi dequeue menjadi O(1) dengan cara menggerakkan pointer head maju tanpa menggeser elemen, namun menciptakan "lubang" atau ruang kosong di awal array yang tidak dapat digunakan kembali, sehingga queue bisa menjadi penuh meskipun masih ada slot kosong di index awal, yang membuat penggunaan memori kurang efisien. Alternatif 3 (Circular Queue - head dan tail berputar) merupakan solusi paling optimal yang menggabungkan keunggulan kedua alternatif sebelumnya dengan menggunakan operasi modulo % 5 untuk membuat array berputar seperti lingkaran, di mana head dan tail dapat kembali ke index 0 setelah mencapai akhir array, sehingga mencapai efisiensi waktu O(1) untuk kedua operasi enqueue dan dequeue sekaligus memaksimalkan penggunaan memori karena seluruh slot array dapat digunakan kembali secara berputar, menjadikannya implementasi yang paling efisien dan direkomendasikan untuk penggunaan praktis dalam aplikasi nyata.

E. Rerferensi

https://dev.to/kauresss/queue-data-structure-in-c-and-operations-256n

https://www.geeksforgeeks.org/cpp/queue-cpp-stl/

https://www.geeksforgeeks.org/dsa/queue-data-structure/