

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 8
PENGENALAN CODE BLOCKS**



Disusun Oleh :

NAMA : Muhamad Ilham Syahid

NIM : 103112400155

Dosen

WAHYU ANDI SAPUTRA

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Queue adalah struktur data linear yang menerapkan prinsip FIFO (First In First Out), yaitu elemen yang pertama masuk akan keluar lebih dahulu. Queue memiliki dua penunjuk utama, yaitu head sebagai penunjuk elemen terdepan dan tail sebagai penunjuk elemen terakhir. Operasi utama pada queue adalah enqueue untuk menambah elemen di bagian belakang dan dequeue untuk menghapus elemen di bagian depan. Queue dapat diimplementasikan menggunakan linked list maupun array, di mana implementasi array dapat dikembangkan menjadi circular queue agar penggunaan memori lebih efisien.

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

Guided 1

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

#define MAX 5

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

// Membuat queue kosong
void createQueue(Queue &Q) {
    Q.head = 0;
    Q.tail = -1;
}

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

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

// Menambahkan elemen (enqueue)
void enqueue(Queue &Q, int x) {
    if (isFullQueue(Q)) {
        cout << "Queue penuh" << endl;
    } else {
```

```

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

// Menghapus elemen (dequeue)
int dequeue(Queue &Q) {
    int x;
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
        return -1;
    } else {
        x = Q.info[Q.head];
        // Geser 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 << Q.head << " - " << Q.tail << "\t | ";
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong";
    } else {
        for (int i = Q.head; i <= Q.tail; i++) {
            cout << Q.info[i] << " ";
        }
    }
    cout << endl;
}

int main() {
    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;
}

```

Screenshots Output

```

PS C:\Pratikum Struktur Data\modul 8 laprak> cd "c:\Pratikum Struktur Data\modul 8 laprak\" ; if ($?) {
g++ queue.cpp -o queue } ; if ($?) { .\queue }
-----
H - T | Queue info
-----
0 - -1 | Queue kosong
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
PS C:\Pratikum Struktur Data\modul 8 laprak>

```

Deskripsi:

Program ini mengimplementasikan struktur data Queue menggunakan array dengan mekanisme Alternatif 1, yaitu head tetap dan tail bergerak, serta menerapkan prinsip FIFO (First In First Out). Queue diinisialisasi dalam keadaan kosong dengan tail = -1, kemudian operasi enqueue digunakan untuk menambahkan data ke bagian belakang queue dengan menaikkan nilai tail, sedangkan operasi dequeue menghapus data terdepan dengan cara menggeser seluruh elemen ke depan karena posisi head tidak berubah. Program juga menyediakan fungsi pengecekan kondisi queue kosong dan penuh, serta fungsi untuk menampilkan isi queue beserta posisi head dan tail, sehingga memudahkan pengguna dalam memahami alur kerja dan perubahan data pada queue.

E. Kesimpulan

Kesimpulannya, program Queue yang dibuat telah berhasil mengimplementasikan konsep Queue berbasis array dengan mekanisme Alternatif 1, di mana head bersifat tetap dan tail bergerak sesuai dengan prinsip FIFO (First In First Out). Operasi enqueue dan dequeue dapat berjalan dengan baik untuk menambah dan menghapus elemen dari antrian, serta kondisi queue kosong dan penuh dapat terdeteksi dengan benar. Meskipun metode ini mudah dipahami dan cocok untuk pembelajaran dasar struktur data, kelemahannya adalah proses pergeseran elemen saat dequeue yang kurang efisien, sehingga untuk penggunaan yang lebih optimal dapat dikembangkan menggunakan metode circular queue.

F. Referensi

Modul Struktur Data – Modul 8: Queue

