LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :
NAMA : Herdian Abdillah Purnomo
NIM : 103112430048

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Queue (Antrian) adalah struktur data linear yang menerapkan prinsip **FIFO (First In, First Out)**, yaitu elemen yang pertama masuk akan menjadi elemen pertama yang keluar. Queue digunakan dalam berbagai aplikasi seperti sistem antrian, buffer pada komunikasi data, scheduling proses pada sistem operasi, dan lain-lain.

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

Guided 1

main.cpp

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

using namespace std;

int main() {
    Queue Q;

    createQueue(Q);
    printInfo(Q);

    cout <<"\n enqueu 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 enqueu 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;
}
```

queue.cpp

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

using namespace std;

void createQueue(Queue &Q) {
    Q.head = 0;
    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)) {
       Q.tail = (Q.tail + 1) % MAX_QUEUE;
       Q.info[Q.tail] = x;
       Q.count++;
    } else {
        cout << "Antrean Penuh! " << endl;
    }
}

int dequeue(Queue &Q) {
    if (!isEmpty(Q)) {
        int x = Q.info[Q.head];
        Q.head = (Q.head + 1) % MAX_QUEUE;
        Q.count--;
        return x;
    } else {
```

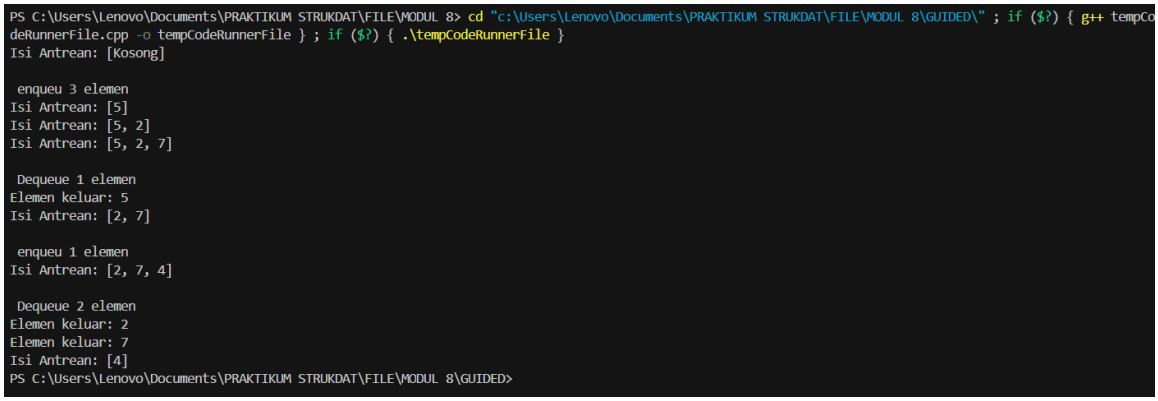
```
        cout << "Antrean Kosong! " << endl;
        return -1;
    }
}

void printInfo(Queue Q) {
    if (isEmpty(Q)) {
        cout << "Isi Antrean: [Kosong]" << endl;
        return;
    }

    cout << "Isi Antrean: [";

    int i = Q.head;
    for (int n = 0; n < Q.count; n++) {
        cout << Q.info[i];
        if (n < Q.count - 1) cout << ", ";
        i = (i + 1) % MAX_QUEUE;
    }

    cout << "]" << endl;
}
```

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
```

Screenshots Output



Deskripsi: Program melakukan simulasi circular queue: membuat queue, menambah elemen dengan enqueue, menghapus elemen dengan dequeue, dan menampilkan isi queue setelah setiap operasi. Pergerakan head–tail menggunakan (index + 1) % MAX_QUEUE.

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

Unguided 1

main.cpp

```
#include "queue.h"
#include "queue.cpp"

void run_test() {
    std::cout << "Hello World" << std::endl;
    std::cout << "H \t T \t : Queue Info" << std::endl;
    std::cout << "----------------------------------------" <<
std::endl;

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

    printInfo(Q);
}

int main() {
    run_test();
    return 0;
}
```

queue.cpp

```cpp
#include "queue.h"

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_SIZE - 1);
}

void printInfo(Queue Q) {
    std::cout << Q.head << " \t " << Q.tail << " \t : ";

    if (isEmptyQueue(Q)) {
        std::cout << "empty queue" << std::endl;
    } else {

        for (int i = Q.head; i <= Q.tail; i++) {
             std::cout << Q.A[i];
             if (i < Q.tail) {
                 std::cout << " ";
             }
        }
        std::cout << std::endl;
    }
}

void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) {
        return;
    }
    if (isEmptyQueue(Q)) {
        Q.head = 0;
        Q.tail = 0;
    } else {
        Q.tail++;
    }
```

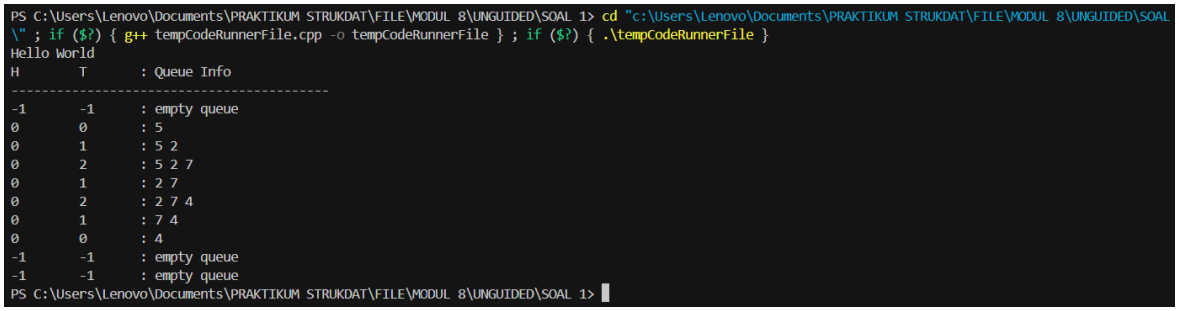
```
    Q.A[Q.tail] = x;
}

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        return 0;
    }
    infotype removed_value = Q.A[Q.head];
    if (Q.head == Q.tail) {
        CreateQueue(Q);
    } else {
        for (int i = Q.head; i < Q.tail; i++) {
            Q.A[i] = Q.A[i + 1];
        }
        Q.tail--;
    }
    return removed_value;
}
```

queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>
#include <cstdlib>

typedef int infotype;
#define MAX_SIZE 5

typedef struct {
    infotype A[MAX_SIZE];
    int head;
    int tail;
} Queue;

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
void printInfo(Queue Q);

bool isFullQueue(Queue Q);
```

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

#endif
```

Screenshots Output



Deskripsi:

Implementasi **linear queue** dengan penggeseran elemen saat dequeue. Hal ini menyebabkan overhead karena elemen harus dipindahkan ke kiri setiap kali ada penghapusan.

Unguided 2

main.cpp

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

void run_test_alt2() {
    std::cout << "Hello world!" << std::endl;
    std::cout << "H    T    : Queue Info" << std::endl;
    std::cout << "----------------------------------------" <<
std::endl;

    Queue Q;
    CreateQueue(Q);

    printInfo(Q);

    for (int i = 1; i <= MAX_SIZE; i++) {
        enqueue(Q, i * 10);
        printInfo(Q);
    }

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

    enqueue(Q, 99);
    printInfo(Q);
}

int main() {
    run_test_alt2();
    return 0;
}
```

queue.cpp

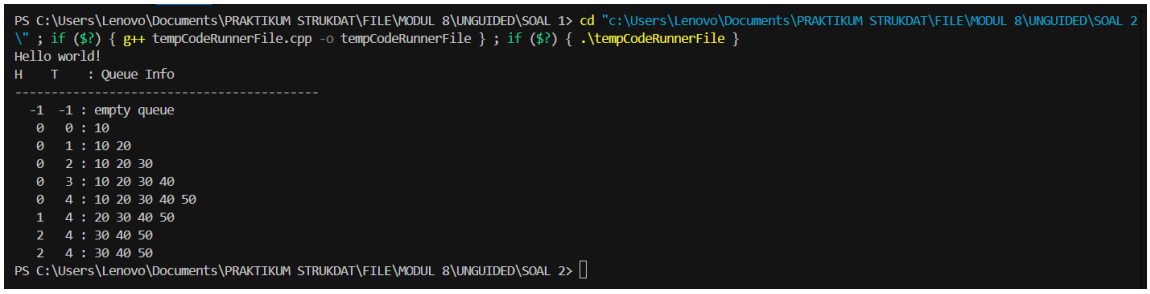
```
#include "queue.h"
#include <iomanip>

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_SIZE - 1); }

void printInfo(Queue Q) {
    std::cout << std::setw(4) << Q.head;
    std::cout << std::setw(4) << Q.tail;
    std::cout << " : ";

    if (isEmptyQueue(Q)) {
        std::cout << "empty queue" << std::endl;
    } else {
        for (int i = Q.head; i <= Q.tail; i++) {
            std::cout << Q.A[i];
            if (i < Q.tail) {
                std::cout << " ";
            }
        }
        std::cout << std::endl;
    }
}

void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) return;
    if (isEmptyQueue(Q)) {
        Q.head = 0;
        Q.tail = 0;
    } else {
        Q.tail++;
    }
    Q.A[Q.tail] = x;
}

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) return 0;
    infotype removed_value = Q.A[Q.head];
    if (Q.head == Q.tail) {
        CreateQueue(Q);
```

```
    } else {
        Q.head++;
    }
    return removed_value;
}
```

queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>
#include <cstdlib>

typedef int infotype;
#define MAX_SIZE 5

typedef struct {
    infotype A[MAX_SIZE];
    int head;
    int tail;
} Queue;

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
void printInfo(Queue Q);

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

#endif
```

Screenshots Output



Deskripsi:

Linear queue tanpa penggeseran. Dequeue hanya menaikkan head, namun queue dianggap penuh saat tail mencapai batas, meskipun masih ada ruang kosong di awal.

Unguided 3

main.cpp

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

void run_test_alt3() {
    std::cout << "Hello world!" << std::endl;
    std::cout << "H    T    : Queue Info" << std::endl;
    std::cout << "----------------------------------------" <<
std::endl;

    Queue Q;
    CreateQueue(Q);

    printInfo(Q);

    for (int i = 1; i < MAX_SIZE; i++) {
        enqueue(Q, i * 10);
        printInfo(Q);
    }

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

    enqueue(Q, 99);
    printInfo(Q);
    enqueue(Q, 88);
    printInfo(Q);

    enqueue(Q, 77);
    printInfo(Q);
}

int main() {
    run_test_alt3();
    return 0;
}
```

queue.cpp

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

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 + 1) % MAX_SIZE == Q.head);
}

void printInfo(Queue Q) {
    std::cout << std::setw(4) << Q.head;
    std::cout << std::setw(4) << Q.tail;
    std::cout << " : ";

    if (isEmptyQueue(Q)) {
        std::cout << "empty queue" << std::endl;
    } else {
        int i = Q.head;
        do {
            std::cout << Q.A[i];
            if (i != Q.tail) {
                std::cout << " ";
            }

            i = (i + 1) % MAX_SIZE;
        } while (i != (Q.tail + 1) % MAX_SIZE);

        std::cout << std::endl;
    }
}

void enqueue(Queue &Q, infotype x) {
    if (isFullQueue(Q)) return;

    if (isEmptyQueue(Q)) {
```

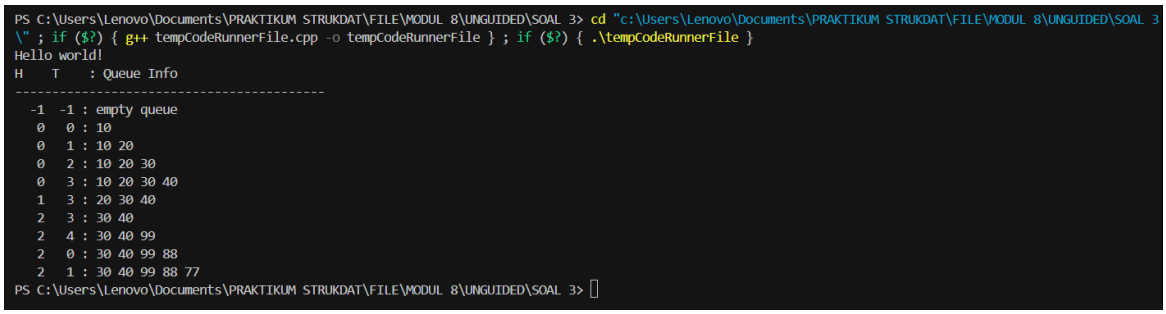
```
        Q.head = 0;
        Q.tail = 0;
    } else {
        Q.tail = (Q.tail + 1) % MAX_SIZE;
    }
    Q.A[Q.tail] = x;
}

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) return 0;

    infotype removed_value = Q.A[Q.head];

    if (Q.head == Q.tail) {
        CreateQueue(Q);
    } else {
        Q.head = (Q.head + 1) % MAX_SIZE;
    }
    return removed_value;
}
```

queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#include <iostream>
#include <cstdlib>

typedef int infotype;
#define MAX_SIZE 5

typedef struct {
    infotype A[MAX_SIZE];
    int head;
    int tail;
} Queue;

void CreateQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
void printInfo(Queue Q);
```

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

#endif
```

Screenshots Output



Deskripsi:

Implementasi **circular queue**. Head dan tail berputar sehingga ruang kosong dapat dipakai kembali. Lebih efisien dibanding linear queue.

E. Kesimpulan

Berdasarkan percobaan pada modul ini, dapat disimpulkan bahwa:

1. **Queue adalah struktur data FIFO** di mana elemen yang pertama masuk adalah yang pertama keluar.
2. **Linear queue** memiliki kelemahan: ruang kosong di depan tidak dapat digunakan kembali setelah beberapa operasi dequeue.
3. **Circular queue** lebih efisien karena memanfaatkan seluruh kapasitas array dengan memutar posisi head dan tail menggunakan operasi modulo.
4. Implementasi queue perlu memperhatikan kondisi-kondisi khusus seperti queue penuh, queue kosong, dan pembaruan head–tail.
5. Fungsi printInfo membantu memvisualisasikan pergerakan data sehingga memudahkan pemahaman konsep queue.

F. Referensi

Raharjo, Budi. 2025. Buku Pemrograman C++ Mudah dan Cepat Menjadi Master C.

Wikipedia contributors. (2024, 8 Mei). C++. Wikipedia, Ensiklopedia Bebas. Diakses pada 2 Oktober 2025, dari https://id.wikipedia.org/wiki/C%2B%2B