LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII

QUEUE

Disusun Oleh :
NAMA : DEVI YULIANA
NIM : 103112400151

Dosen
FAHRUDIN MUKTI WIBOWO
STRUKTUR DATA

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue adalah struktur data yang bekerja dengan prinsip FIFO, di mana elemen yang masuk lebih dahulu akan keluar lebih dahulu. Dalam array, posisi elemen dikendalikan oleh head dan tail. Terdapat tiga mekanisme utama. Alternatif pertama membuat head tetap dan tail bergerak, namun penghapusan elemen membutuhkan pergeseran. Alternatif kedua membuat head dan tail sama-sama bergerak tanpa pergeseran, tetapi dapat menimbulkan kondisi penuh semu. Alternatif ketiga, circular queue, membuat head dan tail berputar sehingga ruang array dapat dimanfaatkan penuh tanpa perlu menggeser elemen.

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

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
```

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];
```

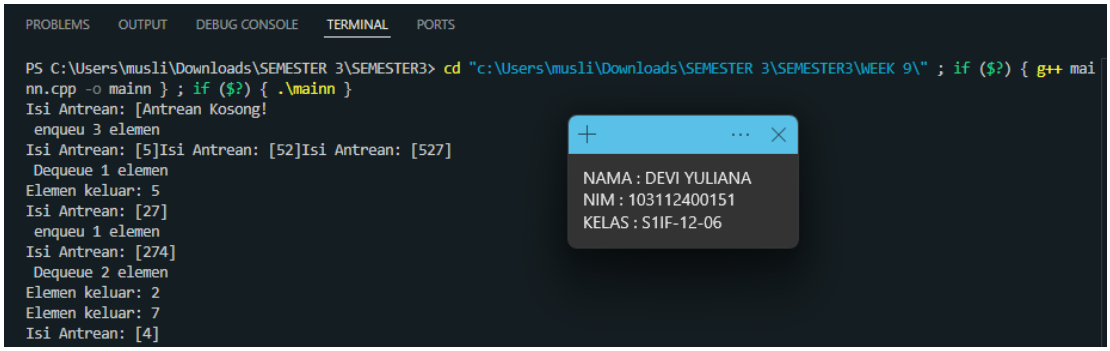
```
        Q.head = (Q.head + 1) % MAX_QUEUE;
        Q.count--;
        return x;
    } else {
        cout << "Antrean Kosong! " << endl;
        return -1;
    }
}

void printInfo(Queue Q) {
    cout << "Isi Antrean: [";
    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 << "]";
    } else {
        cout << "Antrean Kosong!";
    }
}
```

main.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;

}
```

Screenshots Output



Deskripsi :

Program ini memperlihatkan cara kerja queue dengan prinsip FIFO. Data dimasukkan lewat enqueue, diambil lewat dequeue, dan setiap perubahan langsung ditampilkan. Melalui proses ini, terlihat jelas bahwa elemen selalu masuk dari belakang dan keluar dari depan sesuai urutan kedatangannya.

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

Unguided 1

queue.h

```
// queue.cpp  (ALTERNATIF 3: circular queue)
#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) {
    if (isEmptyQueue(Q)) return false;
    return ((Q.tail + 1) % MaxEl) == Q.head;
}

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

    if (isEmptyQueue(Q)) {
        Q.head = 0;
        Q.tail = 0;
```

```
    } else {
        Q.tail = (Q.tail + 1) % MaxEl;
    }

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

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
        return 0;           // nilai dummy
    }

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

    if (Q.head == Q.tail) {
        // setelah ambil elemen terakhir -> kosong
        Q.head = -1;
        Q.tail = -1;
    } else {
        Q.head = (Q.head + 1) % MaxEl;
    }

    return x;
}

void printInfo(Queue Q) {
    if (isEmptyQueue(Q)) {
        cout << "-1 - -1\t| empty queue" << endl;
    } else {
        cout << Q.head << " - " << Q.tail << "\t| ";

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

queue.cpp

```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 == MaxEl - 1;
}

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

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

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

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong\n";
        return 0;              // nilai dummy
    }
```

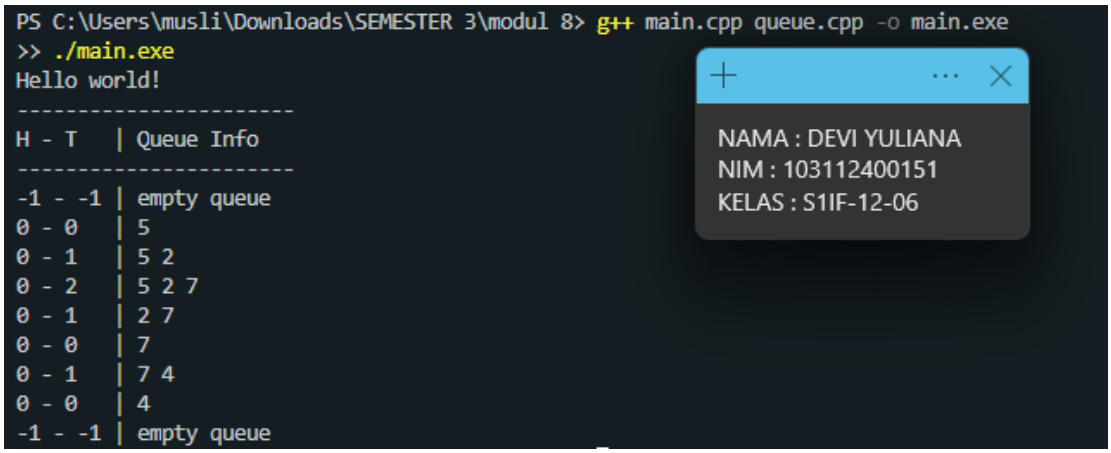
```
    infotype x = Q.info[Q.head];

    if (Q.head == Q.tail) {
        // setelah ambil elemen terakhir -> kosong
        Q.head = -1;
        Q.tail = -1;
    } else {
        // head diam di 0, elemen digeser ke kiri
        for (int i = Q.head + 1; i <= Q.tail; ++i) {
            Q.info[i - 1] = Q.info[i];
        }
        Q.tail--;
    }

    return x;
}

void printInfo(Queue Q) {
    if (isEmptyQueue(Q)) {
        cout << "-1 - -1\t| empty queue" << endl;
    } else {
        cout << Q.head << " - " << Q.tail << "\t| ";
        for (int i = Q.head; i <= Q.tail; ++i) {
            cout << Q.info[i] << " ";
        }
        cout << endl;
    }
}
```

main.cpp

```
#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);
    dequeue(Q);    printInfo(Q);
    enqueue(Q, 4); printInfo(Q);
    dequeue(Q);    printInfo(Q);
    dequeue(Q);    printInfo(Q);

    return 0;
```

Screenshots Output



Deskripsi :

Program ini membuat queue dengan head yang selalu tetap di posisi awal, sedangkan tail bergerak setiap kali data baru dimasukkan. Enqueue dilakukan dengan menambah elemen di posisi tail. Saat dequeue, elemen depan diambil lalu semua elemen digeser ke kiri supaya head tetap di awal. Mekanismenya sederhana, tetapi kurang efisien karena penggeseran terjadi setiap kali dequeue.

Unguided 2

queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H

const int MaxEl = 5;
typedef int infotype;

struct Queue {
    infotype info[MaxEl];   // index 0..4
    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
```

queue.cpp

```cpp
// queue.cpp  (ALTERNATIF 2: head & tail bergerak, non-circular)
#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;
}
```

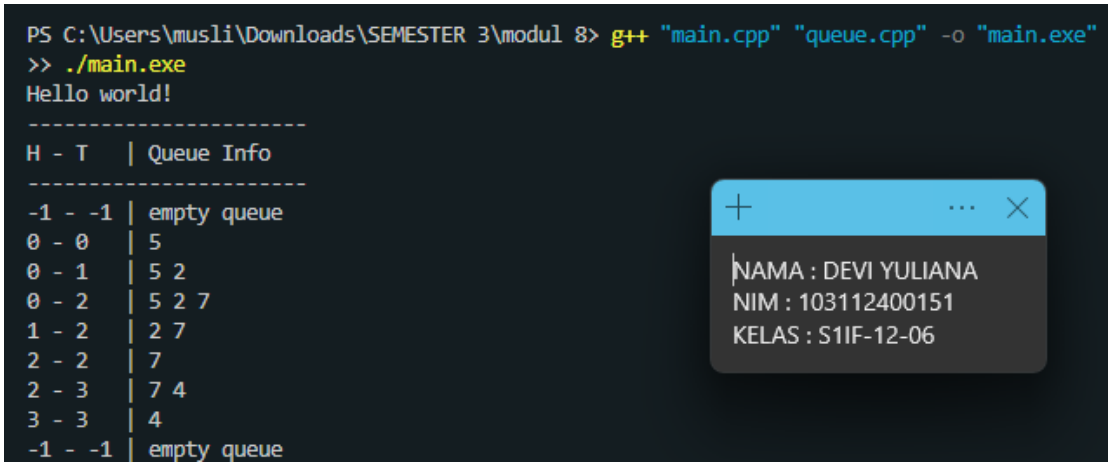
```
bool isFullQueue(Queue Q) {
    return Q.tail == MaxEl - 1;
}

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

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

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

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
        return 0;       // nilai dummy
    }

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

    if (Q.head == Q.tail) {
        // elemen tinggal 1, setelah diambil -> kosong
        Q.head = -1;
        Q.tail = -1;
    } else {
        // head maju satu posisi, tail tetap
        Q.head++;
    }

    return x;
}

void printInfo(Queue Q) {
```

```
    if (isEmptyQueue(Q)) {
        cout << "-1 - -1\t| empty queue" << endl;
    } else {
        cout << Q.head << " - " << Q.tail << "\t| ";
        for (int i = Q.head; i <= Q.tail; ++i) {
            cout << Q.info[i] << " ";
        }
        cout << endl;
    }
}
```

main.cpp

```
#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);
    dequeue(Q);    printInfo(Q);
    enqueue(Q, 4); printInfo(Q);
    dequeue(Q);    printInfo(Q);
    dequeue(Q);    printInfo(Q);

    return 0;
}
```

Screenshots Output



Deskripsi:

Pada program ini, head dan tail sama-sama bisa bergerak. Enqueue membuat tail maju, dan dequeue cukup membuat head maju tanpa perlu menggeser elemen lain. Cara ini lebih efisien daripada alternatif pertama. Kekurangannya, ruang kosong di awal array bisa terbuang, sehingga queue bisa terlihat penuh padahal masih ada tempat kosong.

Unguided 3

queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H

const int MaxEl = 5;
typedef int infotype;

struct Queue {
    infotype info[MaxEl];   // index 0..4
    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
```

queue.cpp

```
// queue.cpp  (ALTERNATIF 3: circular queue)
#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) {
    if (isEmptyQueue(Q)) return false;
    return ((Q.tail + 1) % MaxEl) == Q.head;
}

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

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

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

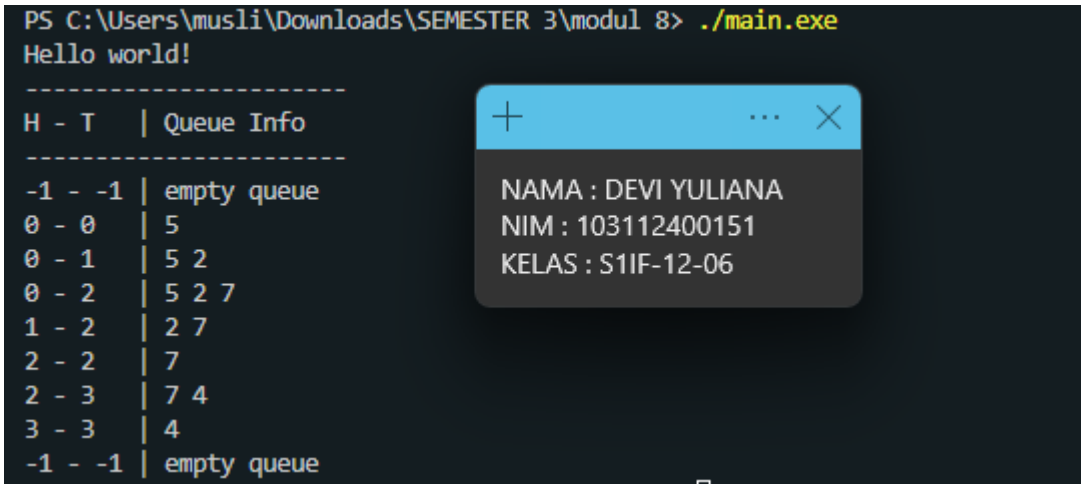
```
}

infotype dequeue(Queue &Q) {
    if (isEmptyQueue(Q)) {
        cout << "Queue kosong" << endl;
        return 0;           // nilai dummy
    }

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

    if (Q.head == Q.tail) {
        // setelah ambil elemen terakhir -> kosong
        Q.head = -1;
        Q.tail = -1;
    } else {
        Q.head = (Q.head + 1) % MaxEl;
    }

    return x;
}

void printInfo(Queue Q) {
    if (isEmptyQueue(Q)) {
        cout << "-1 - -1\t| empty queue" << endl;
    } else {
        cout << Q.head << " - " << Q.tail << "\t| ";

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

main.cpp

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

```cpp
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;
}
```

Screenshots Output



Deskripsi:

Program ini menggunakan sistem melingkar. Ketika mencapai ujung array, head dan tail bisa kembali ke indeks awal. Dengan cara ini, seluruh ruang array termanfaatkan tanpa perlu penggeseran atau masalah penuh semu. Alternatif ini paling efisien dan stabil dibanding dua mekanisme sebelumnya.

D. Kesimpulan

Queue adalah struktur data yang bekerja dengan prinsip FIFO, di mana elemen yang masuk lebih dulu akan keluar lebih dulu. Tiga mekanisme implementasi queue pada array memiliki karakteristik berbeda: alternatif pertama sederhana tetapi kurang efisien karena membutuhkan pergeseran elemen, alternatif kedua lebih cepat namun bisa mengalami kondisi penuh semu, sementara alternatif ketiga menggunakan sistem melingkar yang memanfaatkan ruang secara optimal tanpa pergeseran. Secara keseluruhan, circular queue menjadi metode yang paling efisien dan stabil dibanding dua pendekatan lainnya.

E. Referensi

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY.*