LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII
QUEUE

Disusun oleh :

Junadil Muqorobin (103112400281)

Dosen

Fahrudin Mukti Wibowo S.Kom., M.Eng

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue atau antrian merupakan salah satu bentuk struktur data linear yang menggunakan aturan **FIFO (First In, First Out)**, yaitu elemen yang pertama dimasukkan akan menjadi elemen pertama yang diproses atau dikeluarkan. Prinsip ini membuat queue sangat mirip dengan situasi antrian di dunia nyata, sehingga banyak digunakan dalam sistem komputasi yang memerlukan pengelolaan proses secara berurutan. Dalam implementasi perangkat lunak, operasi dasar pada queue meliputi enqueue untuk menambahkan elemen di bagian belakang antrian, dequeue untuk mengambil elemen dari bagian depan, serta pengecekan kondisi penuh dan kosong. Berbagai penelitian di Indonesia juga mencatat bahwa konsep FIFO mempermudah pengaturan alur layanan dan pemrosesan data yang memerlukan urutan stabil dari awal hingga akhir pemrosesan (Ismail et al., 2024).

Implementasi queue berbasis array lebih mudah dipahami karena indeks sudah terstruktur dan kapasitasnya tetap. Namun, jika elemen sering di-*dequeue*, implementasi dasar (Alternatif 1) memerlukan proses *shifting* agar elemen terdepan selalu berada pada indeks 0, dan hal ini menyebabkan operasi dequeue memiliki kompleksitas O(n). Meski memiliki kelemahan, pendekatan ini masih sering digunakan dalam konteks pembelajaran karena logikanya sederhana dan mudah diobservasi. Sebaliknya, queue berbasis linked list lebih fleksibel karena ukurannya dapat bertambah sesuai kebutuhan dan operasi enqueue maupun dequeue dapat mencapai O(1), meskipun setiap elemen membutuhkan memori tambahan berupa pointer yang membuat overhead memori lebih besar (Syntax Literate, 2022).

Selain itu, terdapat dua pendekatan populer lainnya dalam implementasi queue array: Alternatif 2 (head dan tail bergerak tanpa melakukan shift) dan Alternatif 3 (circular queue). Pada Alternatif 2, posisi head dan tail akan terus bergeser ke kanan mengikuti penambahan atau pengurangan elemen. Pendekatan ini menghindari proses shifting, sehingga operasi lebih efisien. Namun, karena array memiliki ukuran tetap, area kosong di bagian depan tidak dapat dimanfaatkan kembali tanpa dilakukan reset. Untuk mengatasi kelemahan tersebut, Alternatif 3 atau circular queue memperkenalkan mekanisme pergerakan head dan tail secara melingkar dengan operasi modulo. Pendekatan ini memungkinkan seluruh slot array digunakan secara optimal tanpa perlu melakukan shifting atau reset manual. Circular queue banyak direkomendasikan dalam penelitian karena efisien dan cocok untuk buffer tetap, seperti pada sistem antrean layanan dan sistem penjadwalan proses (Prayitno, 2024).

Penggunaan struktur data queue memiliki relevansi yang luas dalam berbagai aplikasi modern. Sistem antrian daring, manajemen layanan pelanggan, penjadwalan proses pada sistem operasi, hingga arsitektur jaringan memanfaatkan mekanisme antrian untuk menjaga keteraturan data yang masuk dan keluar. Berbagai studi kasus di Indonesia menunjukkan bahwa pemilihan jenis implementasi queue sangat ditentukan oleh kebutuhan efisiensi ruang dan waktu, serta jumlah data yang diproses secara simultan. Dalam sistem yang terus beroperasi dengan beban data besar dan konstan, circular queue lebih banyak dipilih karena mampu menjaga performa tanpa menghabiskan ruang yang tidak terpakai, sementara untuk aplikasi kecil atau pembelajaran dasar, queue linear dengan shifting masih dianggap cukup efektif (Apriliyanto, 2025).

B. Guided

1. Implementasi Queue

Source code queue.h

```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
```

Source code queue.cpp

```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++;
    }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 Queue: [";
    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;
}
```

Source code main.cpp

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

using namespace std;

int main(){
```

```
    Queue Q;

    createQueue(Q);
    printInfo(Q);

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

Screenshoot Output:

Deskripsi:

Program ini mengimplementasikan struktur data Queue menggunakan array berukuran tetap dengan mekanisme circular queue. Posisi head dan tail bergerak secara melingkar menggunakan operasi modulo sehingga seluruh slot array dapat dimanfaatkan tanpa penggantian elemen. Program berjalan dengan langkah kerja sebagai berikut:

- Source code queue.h

  File ini berisi deklarasi tipe dadta abstrak queue dan fungsi-fungsi yang digunakan, terdiri dari info[MAX_QUEUE] yaitu array untuk menyimpan elemen queue, head adalah indeks elemen paling depan, tail indeks posisi penambahan elemen, dan count jumlah elemen di dalam queue saat ini. Selain itu juga terdapat definisi abstrak dari fungsi yang digunakan.

- Source code queue.cpp

  File ini berisi penerapan prototype fungsi yang ada pada ADT queue, sebagai berikut:

  - createQueue(Queue &Q) untuk menginisialisasi queue kosong dengan head = 0, tail = 0, dan count = 0. Pada tahap ini, queue belum memiliki elemen apa pun.
  - isEmpty(Queue Q) untuk mengembalikan nilai benar saat count == 0, artinya queue tidak memiliki elemen.
  - isFull(Queue Q) untuk mengembalikan nilai benar ketika jumlah elemen mencapai batas kapasitas (count == MAX_QUEUE).

- enqueue(Queue &Q, int x) untuk menambahkan elemen baru ke posisi tail. Setelah memasukkan nilai, indeks tail digeser maju secara melingkar menggunakan (tail + 1) % MAX_QUEUE, kemudian count ditambah satu. Bila queue penuh, muncul pesan “Antrean Penuh!”.
- dequeue(Queue &Q) untuk mengambil elemen pada posisi head. Setelah elemen diambil, head digeser maju menggunakan (head + 1) % MAX_QUEUE, dan count dikurangi. Jika queue kosong, fungsi menampilkan pesan “Antrean Kosong!” dan mengembalikan -1.
- printInfo(Queue Q) untuk menampilkan seluruh elemen queue dalam format [...]. Pergerakan indeks menggunakan pola circular, sehingga traversal dilakukan dari posisi head sebanyak count langkah.

- Source code main.cpp

  File ini berisi alur utama program. Program dimulai dengan membuat sebuah objek Queue dan kemudian memanggil createQueue untuk menginisialisasinya. Setelah itu, dilakukan serangkaian operasi:
  - Menampilkan kondisi queue awal
  - Melakukan **enqueue** tiga elemen (5, 2, 7) dan menampilkan hasilnya
  - Melakukan **dequeue** satu elemen
  - Melakukan **enqueue** satu elemen (4)
  - Melakukan **dequeue** dua elemen terakhir

- Menampilkan kondisi queue setelah setiap perubahan

C. Unguided

1. Membuat ADT Queue Menggunakan Array

Source code queue.h

```h
#ifndef QUEUE_H
#define QUEUE_H

#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
```

Source code 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 == 4);
}

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 << "empty queue" << endl;
        return -999;
    }

    infotype temp = 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 = -1;
        Q.tail = -1;
    }

    return temp;
}

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

    if (isEmptyQueue(Q)) {
```

```
        cout << "empty queue" << endl;
        return;
    }

    for (int i = Q.head; i <= Q.tail; i++) {
        cout << Q.info[i] << " ";
    }

    cout << endl;
}
```

Source code 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;
}
```

Source code queueDua.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 == 4);
}

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 -999;
    }

    infotype temp = Q.info[Q.head];
    Q.head++;

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

    return temp;
}

void printInfo(Queue Q) {
    cout << " " << Q.head << " - " << Q.tail << "\t | ";
```

```
    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    for (int i = Q.head; i <= Q.tail; i++) {
        cout << Q.info[i] << " ";
    }
    cout << endl;
}
```

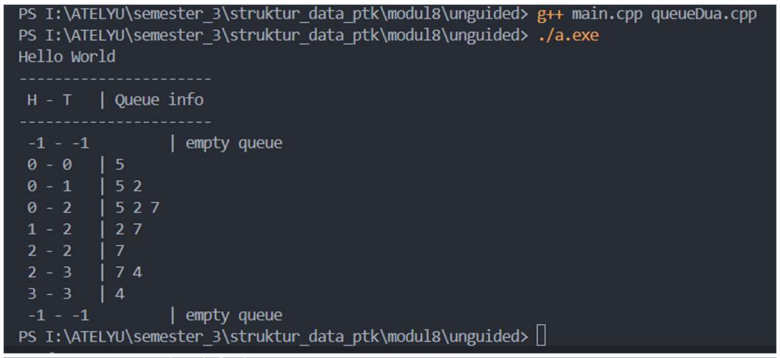
Source code queueTiga.cpp

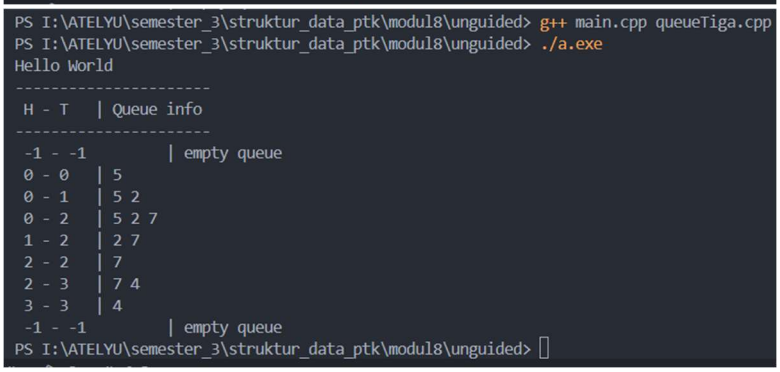
```
#include "queue.h"

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 (!isEmptyQueue(Q) && ((Q.tail + 1) % 5 == 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) % 5;
    }

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

infotype dequeue(Queue &Q) {
```

```
    if (isEmptyQueue(Q)) {
        cout << "queue kosong" << endl;
        return -999;
    }

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

    if (Q.head == Q.tail) {
        Q.head = -1;
        Q.tail = -1;
    } else {
        Q.head = (Q.head + 1) % 5;
    }

    return temp;
}

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

    if (isEmptyQueue(Q)) {
        cout << "empty queue" << endl;
        return;
    }

    int i = Q.head;
    while (true) {
        cout << Q.info[i] << " ";
        if (i == Q.tail) break;
        i = (i + 1) % 5;
    }
    cout << endl;
}
```

Screenshot Output:

```
PS I:\ATELYU\semester_3\struktur_data_ptk\modul8\unguided> g++ main.cpp queue.cpp
PS I:\ATELYU\semester_3\struktur_data_ptk\modul8\unguided> ./a.exe
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 I:\ATELYU\semester_3\struktur_data_ptk\modul8\unguided>
```





Deskripsi:

Program diatas adalah implementasi tiga variasi queue menggunakan array statis berukuran lima elemen. Pada soal 1 elemen-elemen digeser setiap kali dilakukan operasi *dequeue,* kemudian pada soal 2 head dan tail bergerak maju tanpa pergeseran data, dan pada soal 3 *queue* berbentuk melingkar (circular queue), sehingga head dan tail dapat berputar kembali ke indeks awal ketika mencapai batas array. Program berjalan dengan alur kerja sebagai berikut:

- Unguided soal 1
  - createQueue() untuk menginisialisasi queue dalam keadaan kosong dengan head = -1 dan tail = -1.

- isEmptyQueue() yaitu mengecek apakah queue kosong berdasarkan kondisi head dan tail.
- isFullQueue() untuk mengecek apakah queue penuh dengan melihat apakah tail == 4.
- Enqueue() berjalan dengan cara jika queue kosong maka head dan tail menjadi 0, jika tidak kosong maka tail bertambah dan data disimpan pada posisi info[tail]
- Dequeue() berjalan dengan cara mengambil elemen pertama, kemudian menggeser seluruh elemen ke kiri dan tail dikurangi satu jika tail < 0 maka queue kembali kosong.
- printInfo() digunakan untuk menampilkan nilai head, tail, dan isi queue dari indeks 0 hingga tail.

- Unguided soal 2
  - createQueue() unutuk mengatur head dan tail ke posisi -1 atau kosong
  - isEmptyQueue() queue kosong bila head dan tail = -1
  - isFullQueue() queue penuh bila tail berada pada indeks terakhir
  - enqueue() jika kosong maka head = tail = 0 dan jika tidak maka tail bertambah serta tidak ada pergeseran.
  - Dequeue() elemen yang dihapus adalalh info[head] dengan head maju datu posisi

dan jika head melewati tail maka queue kosong Kembali

- printInfo() digunakan nutuk menampilkan elemen dari indeks head hingga tail.

- Unguided soal 3
  - createQueue() untuk menginisialisasi queue kosong.
  - isEmptyQueue() yaitu Queue kosong jika head dan tail = -1.
  - isFullQueue() yaitu Queue penuh jika (tail + 1) % 5 == head. Artinya queue sudah berputar dan ruang habis.
  - Enqueue() digunakan jika kosong →head = tail = 0, jika tidak kosong → tail berputar dengan (tail + 1) %5 dan kemudian data dimasukkan pada posisi tail terbaru.
  - printInfo() digunakan untuk menampilkan semua elemen dengan pergerakan indeks melingkar atau modulo.

D. Kesimpulan

Pada praktikum ini, saya dapat mengimplementasikan tiga variasi mekanisme *queue* menggunakan struktur data array statis, yaitu Alternatif 1 (shifting), Alternatif 2 (head dan tail bergerak secara linear), serta Alternatif 3 (circular queue). Setiap metode memiliki karakteristik dan perilaku berbeda dalam menangani operasi enqueue dan dequeue, sehingga memberikan gambaran yang lebih luas mengenai cara kerja antrean pada level memori. Melalui perbandingan ketiga implementasi tersebut, terlihat

bahwa shifting pada Alternatif 1 kurang efisien karena memindahkan data setiap kali penghapusan elemen. Alternatif 2 lebih baik karena menghilangkan shifting, namun masih memiliki keterbatasan ruang ketika slot di awal array kosong. Circular queue pada Alternatif 3 menjadi solusi paling optimal karena mampu memanfaatkan seluruh kapasitas array dengan mekanisme perputaran head dan tail. Secara keseluruhan, praktikum ini membantu memperkuat pemahaman mengenai konsep FIFO serta menunjukkan bagaimana perbedaan pendekatan implementasi dapat memengaruhi efisiensi struktur data.

E. Referensi

Apriliyanto, E. (2025). *Queue Optimization Using Heuristic-Based Decision*. E-Jurnal SINUS.

Ismail, J., Gea, M. N., Satria, H., Lubis, H. T., Prasetya, H., Hanani, J., & Sihombing, R. (2024). *Implementasi Algoritma FIFO terhadap Sistem Antrian Pasien di Rumah Sakit Berbasis Web Online*. Jurnal Energi, Sistem, dan Komputasi Elektronik.

Prayitno, S. (2024). *Raw Material Truck's Queue System Optimization (Optimasi Sistem Antrian Truk Bahan Baku)*. PELS: Jurnal Teknik.

Syntax Literate. (2022). *Pemanfaatan Algoritma Queue dalam Pengaturan Penjualan pada Usaha JC (Jelly Corner)*. Syntax Literate: Jurnal Informatika.