LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VIII
QUEUE

Disusun Oleh :

NAMA : HILMI HAKIM RAMADANI

NIM : 103112430016

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Queue adalah struktur data yang menggunakan prinsip FIFO (First In First Out), di mana elemen pertama yang dimasukkan ke dalam queue akan menjadi elemen pertama yang dikeluarkan.

Prinsip ini mirip dengan antrean pada umumnya, seperti antrean di kasir atau saat mencetak dokumen. Dalam pemrograman, queue efektif untuk mengatur data atau task yang diproses secara berurutan, memastikan semua operasi dijalankan secara efisien.

Queue juga dapat digunakan dalam berbagai bentuk, termasuk simple queue, circular queue, priority queue, dan double-ended queue (deque), masing-masing dengan karakteristik dan kegunaan khusus sesuai dengan kebutuhan aplikasi.

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

Guided

#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 Denqueue 1 elemen" << endl;

    cout << "elemen keluar:" << dequeue(Q) << endl;
    printInfo(Q);

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

    cout << "\n Denqueue 2 elemen" << endl;
```

```
    cout << "elemen keluar:" << dequeue(Q) << endl;

    cout << "elemen keluar:" << dequeue(Q) << endl;
    printInfo(Q);

    return 0;
}
```

#queue.cpp

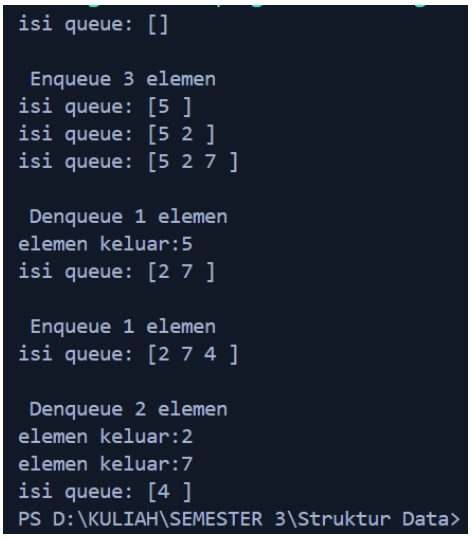
```
#include "queue.h"
#include <iostream>

using namespace std;

void createQueue(Queue &Q)
{
    Q.head = 0;
    Q.tail = 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;
}
```

#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 ini mengimplementasikan struktur data Queue (antrian) berbasis array dengan konsep circular queue menggunakan bahasa C++. Queue direpresentasikan oleh array info dengan kapasitas maksimum (MAX_QUEUE), serta tiga variabel utama yaitu head untuk menunjuk elemen terdepan, tail untuk posisi penambahan elemen, dan count untuk menyimpan jumlah elemen dalam queue. Operasi yang disediakan meliputi enqueue untuk menambah elemen ke belakang antrian, dequeue untuk menghapus dan mengambil elemen dari depan antrian, isEmpty dan isFull untuk mengecek kondisi queue, serta printInfo untuk menampilkan isi queue. Pada fungsi main, program mendemonstrasikan proses enqueue dan dequeue secara bertahap sehingga terlihat bahwa queue bekerja dengan prinsip FIFO (First In First Out), di mana elemen yang pertama masuk akan menjadi elemen pertama yang keluar.

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

Soal1

#main.cpp

```
#include <iostream>
#include "queue.h"
#include "queue.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);
    enqueue(Q, 4);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);
    dequeue(Q);
    printInfo(Q);

    return 0;
}
```

#queue.cpp

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

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_QUEUE - 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 -1;
    }

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

    for (int i = Q.head; i < Q.tail; i++)
    {
        Q.info[i] = Q.info[i + 1];
    }

    Q.tail--;

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

    return x;
}

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

    if (isEmptyQueue(Q))
```

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

```
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 - 2  | 2 7 4
0 - 1  | 7 4
0 - 0  | 4
PS D:\KULIAH\SEMESTER 3\Struktur Data>
```

Soal 2

- Menambah code di queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#define MAX_QUEUE 5
typedef int infotype;

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

- Menambah code di queue.cpp

```
#include "queue.h"
#include <iostream>
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 + 1) % MAX_QUEUE == 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) % MAX_QUEUE;
   }

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

infotype dequeue(Queue &Q)
{
   if (isEmptyQueue(Q))
```

```
    {
        cout << "Queue kosong" << endl;
        return -1;
    }

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

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

    return x;
}

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

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

- Menambah code di main.cpp

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

```
#include "queue.cpp"
using namespace std;

int main()
{
   cout << "Hello world!" << 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);

   return 0;
}
```

- Output

```
Hello world!
-1 - -1 | empty queue
0 - 0 | 5
0 - 1 | 5 2
0 - 2 | 5 2 7
1 - 2 | 2 7
1 - 3 | 2 7 4
2 - 3 | 7 4
3 - 3 | 4
PS D:\KULIAH\SEMESTER 3\Struktur Data>
```

Soal 3

- Tambahkan code di queue.h

```
#ifndef QUEUE_H
#define QUEUE_H

#define MAX_QUEUE 5
typedef int infotype;
```

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

- Tambahkan code di queue.cpp

```
#include "queue.h"
#include <iostream>
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 + 1) % MAX_QUEUE == 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) % MAX_QUEUE;
    }

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

infotype dequeue(Queue &Q)
{
    if (isEmptyQueue(Q))
    {
        cout << "Queue kosong" << endl;
        return -1;
    }

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

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

    return x;
}

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

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

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

- Tambahkan code di main.cpp

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

int main()
{
    cout << "Hello world!" << 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);

    return 0;
```

```
}
```

- Output

```
Hello world!
-1 - -1 | empty queue
0 - 0 | 5
0 - 1 | 5 2
0 - 2 | 5 2 7
1 - 2 | 2 7
1 - 3 | 2 7 4
2 - 3 | 7 4
3 - 3 | 4
PS D:\KULIAH\SEMESTER 3\Struktur Data>
```

Deskripsi:

Implementasi ADT Queue terdiri dari tiga alternatif. **Alternatif 1** menggunakan array dengan head tetap di indeks 0 dan tail bergerak, sehingga dequeue memerlukan pergeseran data dan kurang efisien meskipun mudah dipahami serta menunjukkan konsep FIFO dengan jelas. **Alternatif 2** dan **Alternatif 3** menerapkan circular array, di mana head dan tail bergerak secara melingkar tanpa pergeseran data, sehingga setiap operasi enqueue dan dequeue berjalan lebih efisien dengan kompleksitas O(1). Pendekatan circular ini lebih optimal dalam pemanfaatan memori dan banyak digunakan pada sistem antrian nyata seperti buffer data, antrian proses, dan sistem operasi.

D. Kesimpulan

Berdasarkan praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data Queue merupakan struktur data linear yang bekerja dengan prinsip FIFO (First In First Out), di mana elemen yang pertama masuk akan menjadi elemen pertama yang keluar. Melalui implementasi ADT Queue berbasis array, mahasiswa dapat memahami cara kerja operasi dasar seperti enqueue, dequeue, pengecekan kondisi kosong dan penuh, serta penampilan isi queue.

Praktikum ini juga menunjukkan perbedaan tiga mekanisme implementasi queue, yaitu

Alternatif 1 dengan head tetap dan tail bergerak yang sederhana namun kurang efisien karena memerlukan pergeseran data, serta Alternatif 2 dan Alternatif 3 yang menggunakan konsep circular queue sehingga lebih efisien dalam penggunaan waktu dan memori karena tidak memerlukan pergeseran data. Dengan demikian, penerapan circular queue lebih direkomendasikan untuk sistem nyata yang membutuhkan pengelolaan antrian secara optimal, seperti buffer data, antrian proses, dan sistem operasi.

E. Referensi

RevoUpedia. 2025, 21 Desember. Stack. Diakses pada 21 Desember. Dari https://www.revou.co/kosakata/queue