LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL 8
QUEUE

Disusun Oleh :
NAMA : LOH SUWARGI NITIS HAMENGKU BINTANG
NIM : 103112400116

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Queue adalah struktur data yang memiliki peran besar dalam pemrograman karena menggunakan pola FIFO dalam mengatur urutan data. Dalam praktiknya, terutama pada sistem embedded dan aplikasi multi-thread, circular queue menjadi pilihan utama berkat efisiensi pengelolaan memorinya. Circular queue bekerja dengan mengembalikan posisi head dan tail ke awal buffer setelah mencapai ujung, sehingga tidak ada ruang yang terbuang. Pendekatan ini juga digunakan pada sistem komputasi dalam kendaraan, di mana antrian melingkar mendukung komunikasi antar thread dengan keterlambatan yang rendah.

Pada teknologi komputasi modern seperti GPU, circular buffer queue dipilih sebagai solusi optimal untuk antrian bounded yang menuntut throughput tinggi. Berdasarkan penelitian "A Fast and Scalable Concurrent FIFO Queue on GPU", penggunaan ring buffer memungkinkan operasi enqueue dan dequeue berlangsung konsisten pada waktu konstan. Manajemen head dan tail dengan operasi modulo mengeliminasi kebutuhan alokasi ulang memori dan meningkatkan kinerja paralel dengan latensi yang kecil.

Pengembangan lebih lanjut melalui teknik seperti BBQ (Block-based Bounded Queue) menunjukkan bahwa konsep antrian melingkar dapat ditingkatkan. Dengan membagi buffer menjadi blok-blok terpisah, potensi benturan pada operasi baca dan tulis dapat diminimalisir. Metode ini memberikan performa lebih baik pada skenario yang melibatkan banyak thread yang mengakses queue secara simultan.

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

Guided 1

```
#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
```

```
#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;
}
```

```
#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;

}
```

Screenshots Output

```
PS C:\semester3\struktur data\modul8\guided1> cd "c:\semester3\struktur data\modul8\gui
if ($?) { .\main }
isi Queue: []

 Enqueue 3 elemen
isi Queue: [5 ]
isi Queue: [5 2 ]
isi Queue: [5 2 7 ]

 Dequeue 1 elemen
Elemen keluar: 5
isi Queue: [2 7 ]

 Enqueue 1 elemen
isi Queue: [2 7 4 ]

 Dequeue 2 elemen
Elemen keluar: 2
Elemen keluar: 7
isi Queue: [4 ]
PS C:\semester3\struktur data\modul8\guided1>
```

Deskripsi:Program ini mengimplementasikan struktur data Queue (antrian) menggunakan konsep circular array. Queue memiliki kapasitas maksimum 5 elemen. Program menyediakan operasi dasar seperti membuat antrian (createQueue), mengecek apakah antrian kosong atau penuh, menambah elemen ke antrian (enqueue), menghapus elemen dari antrian (dequeue), serta menampilkan isi antrian (printInfo).

Di fungsi main, program melakukan beberapa percobaan: mengisi antrian dengan beberapa nilai, menampilkan hasilnya, menghapus elemen, kemudian menambahkan elemen lagi untuk menunjukkan cara kerja antrian melingkar. Hasil keluaran menunjukkan perubahan kepala (head) dan ekor (tail) sesuai operasi yang dilakukan.

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

Unguided 1

```
typedef int infotype;


typedef struct {
    infotype info[5];
    int head;
    int tail;
} Queue;


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);
```

```
#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 && Q.tail == -1);
}

bool isFullQueue(Queue Q){
   return (Q.tail == 4);
}

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 -1;
   }

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

```
    if(Q.head == Q.tail){
        Q.head = Q.tail = -1;
    } else {
        for(int i = Q.head; i < Q.tail; i++){
            Q.info[i] = Q.info[i+1];
        }
        Q.tail--;
    }
    return x;
}

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

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

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

```
using namespace std;

int main() {
    cout << "Hello World" << endl;
    cout << "--------------------" << endl;
    cout << "H - T | Queue Info" << endl;
    cout << "--------------------" << endl;

    Queue Q;
    createQueue(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

```
PS C:\semester3\struktur data\modul8\soal1> ./program
Hello World
--------------------
H - T | Queue Info
--------------------
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 C:\semester3\struktur data\modul8\soal1>
```

Deskripsi: Pada program queue ini model linear diterapkan untuk menggambarkan alur antrian sederhana. Penambahan elemen dilakukan dari belakang dengan menaikkan nilai tail. Konsep ini mudah digunakan karena tidak memerlukan perhitungan khusus pada indeks maupun penanganan overflow yang kompleks.

Untuk proses penghapusan, seluruh isi array harus digeser setelah elemen depan dihapus. Model seperti ini memperlihatkan bagaimana struktur antrian mempertahankan urutan elemen melalui perpindahan indeks. Pendekatan linear ini menjadi fondasi sebelum mempelajari queue yang lebih efisien seperti circular queue.

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

Unguided 2

```
#ifndef QUEUE_H
#define QUEUE_H

struct Queue {
    int data[5];
    int head;
    int tail;
};

void createQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
bool isFullQueue(Queue Q);
```

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

#endif
```

```
#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 && Q.tail == -1);
}

bool isFullQueue(Queue Q) {
    return ((Q.tail + 1) % 5 == Q.head);
}

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

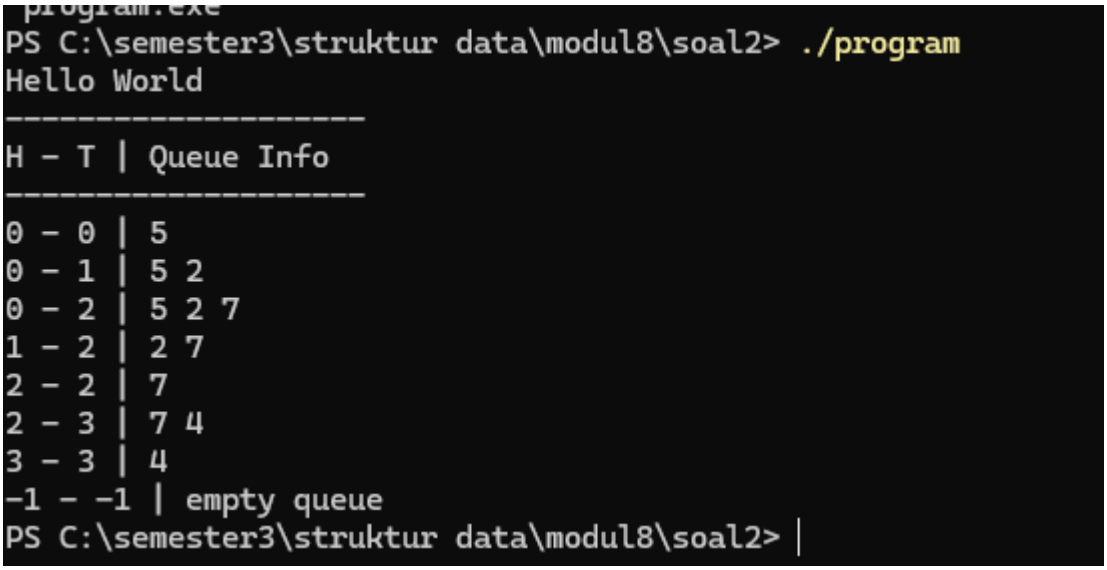
```
    if (isEmptyQueue(Q)) {
        Q.head = 0;
        Q.tail = 0;
    } else {
        Q.tail = (Q.tail + 1) % 5;
    }

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

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

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

void printQueue(Queue Q) {
    if (isEmptyQueue(Q)) {
        cout << Q.head << " - " << Q.tail << " | empty queue" << endl;
        return;
    }
```

```
    cout << Q.head << " - " << Q.tail << " | ";
    int i = Q.head;

    while (true) {
        cout << Q.data[i] << " ";
        if (i == Q.tail) break;
        i = (i + 1) % 5;
    }

    cout << endl;
}
```

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

int main() {
    cout << "Hello World" << endl;
    cout << "--------------------" << endl;
    cout << "H - T | Queue Info" << endl;
    cout << "--------------------" << endl;

    Queue Q;
    createQueue(Q);

    enqueue(Q, 5); printQueue(Q);
    enqueue(Q, 2); printQueue(Q);
```

```
    enqueue(Q, 7); printQueue(Q);

    dequeue(Q); printQueue(Q);
    dequeue(Q); printQueue(Q);

    enqueue(Q, 4); printQueue(Q);

    dequeue(Q); printQueue(Q);
    dequeue(Q); printQueue(Q);

    return 0;
}
```

Screenshots Output



Deskripsi: Kode queue dipisahkan ke dalam tiga file utama: header, implementasi, dan program utama. File header digunakan untuk mendefinisikan struktur queue dan prototipe fungsi, sehingga fungsi ini dapat dipanggil dari file lain. Hal ini menciptakan batas yang jelas antara definisi dan implementasi.

Sementara itu, file queue.cpp memuat seluruh logika fungsi seperti enqueue, dequeue, dan pengecekan kondisi antrian. Program utama hanya perlu memanggil fungsi-fungsi tersebut tanpa menulis ulang implementasinya. Teknik pemisahan ini menghasilkan kode

yang lebih rapi dan terstruktur.

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

Unguided 3

```
#ifndef QUEUE_H
#define QUEUE_H

struct Queue {
    int data[5];
    int head;
    int tail;
    int count;
};

void createQueue(Queue &Q);
bool isEmptyQueue(Queue Q);
bool isFullQueue(Queue Q);
void enqueue(Queue &Q, int x);
void dequeue(Queue &Q);
void printQueue(Queue Q);

#endif
```

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

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

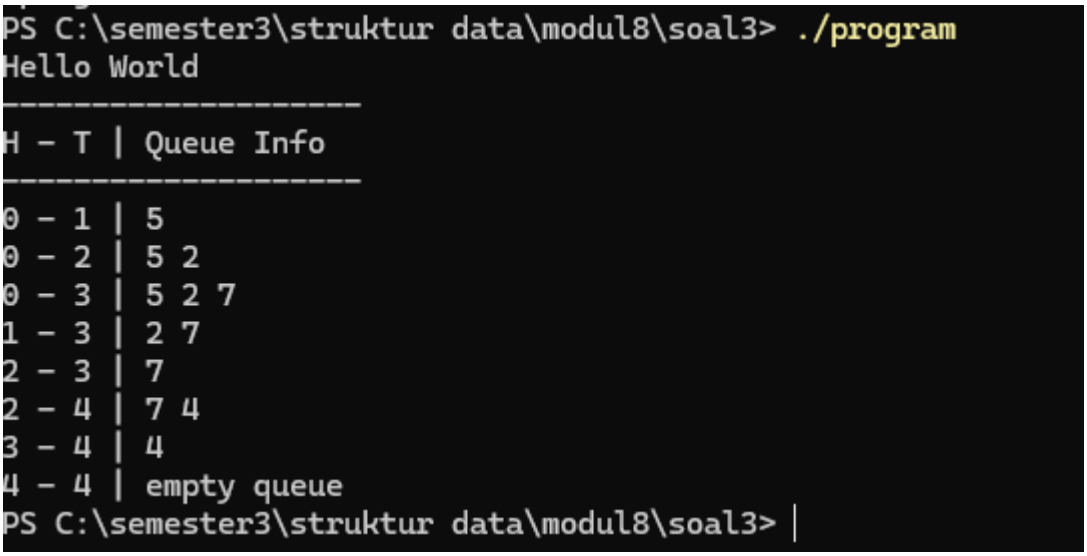
```
    Q.tail = 0;
    Q.count = 0;
}

bool isEmptyQueue(Queue Q) {
    return (Q.count == 0);
}

bool isFullQueue(Queue Q) {
    return (Q.count == 5);
}

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

    Q.data[Q.tail] = x;
    Q.tail = (Q.tail + 1) % 5;
    Q.count++;
}

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

    Q.head = (Q.head + 1) % 5;
```

```
    Q.count--;
}

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

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

    int i = Q.head;
    int printed = 0;

    while (printed < Q.count) {
        cout << Q.data[i] << " ";
        i = (i + 1) % 5;
        printed++;
    }
    cout << endl;
}
```

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

int main() {
```

```
    cout << "Hello World" << endl;
    cout << "--------------------" << endl;
    cout << "H - T | Queue Info" << endl;
    cout << "--------------------" << endl;

    Queue Q;
    createQueue(Q);

    enqueue(Q, 5); printQueue(Q);
    enqueue(Q, 2); printQueue(Q);
    enqueue(Q, 7); printQueue(Q);

    dequeue(Q); printQueue(Q);
    dequeue(Q); printQueue(Q);

    enqueue(Q, 4); printQueue(Q);

    dequeue(Q); printQueue(Q);
    dequeue(Q); printQueue(Q);

    return 0;
}
```

Screenshots Output



Deskripsi: Circular queue memungkinkan pergerakan head dan tail secara melingkar dalam array. Ini berarti ketika tail mencapai posisi akhir, indeks akan kembali ke awal selama masih ada ruang. Pendekatan ini menyelesaikan masalah pemborosan ruang yang terjadi pada queue linear.

Dengan model melingkar, operasi dasar queue tetap mengikuti konsep FIFO tanpa perlu melakukan shifting. Struktur ini lebih efisien karena setiap operasi cukup mengubah indeks, bukan seluruh elemen. Circular queue banyak digunakan ketika sistem membutuhkan antrian cepat dengan memori statis.

G. Kesimpulan

Latihan queue ini membawa kita memahami bagaimana antrian bekerja dari tahap paling dasar hingga implementasi paling efisien. Pada awalnya (nomor 1), antrian linear hanya mampu menambah dan menghapus data dengan cara sederhana, tetapi meninggalkan ruang kosong yang tidak dapat digunakan kembali. Latihan berikutnya (nomor 2) memperbaiki sebagian masalah tersebut dengan memungkinkan pergerakan *head* sehingga ruang tidak terbuang. Akhirnya, nomor 3 memperkenalkan circular queue, yang membuat pergerakan *head* dan *tail* melingkar sehingga seluruh kapasitas dapat digunakan tanpa batasan linear. Dari rangkaian latihan ini dapat disimpulkan bahwa circular queue memberikan pemanfaatan memori terbaik dan bekerja paling efisien di antara semua metode yang dicoba.

H. Referensi

Grammatikakis, M. D., Ninidakis, S., Kornaros, G., Bakoyiannis, D., Mouzakitis, N., & Staridas, A. (2023). Managing Concurrent Queues for Efficient In-Vehicle Gateways. ***J. Commun.***, ***18***(5), 333-339.

Polak, M. S. H., Troendle, D. A., & Jang, B. (2025). Boundary-Aware Concurrent Queue: A Fast and Scalable Concurrent FIFO Queue on GPU Environments. ***Applied Sciences***, ***15***(4), 1834.

Wang, J., Behrens, D., Fu, M., Oberhauser, L., Oberhauser, J., Lei, J., ... & Chen, H. (2022). {BBQ}: A Block-based Bounded Queue for Exchanging Data and Profiling. In ***2022 USENIX Annual Technical Conference (USENIX ATC 22)*** **(pp. 249-262).**