

**LAPORAN PRAKTIKUM  
STRUKTUR DATA**

**MODUL VIII  
QUEUE**



**Disusun Oleh :**  
NAMA : Azzahra Farelika Esti Ning Tyas  
NIM : 103112430023

**Dosen**  
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA  
FAKULTAS INFORMATIKA  
TELKOM UNIVERSITY PURWOKERTO  
2025**

## A. Dasar Teori

Queue atau antrian merupakan salah satu struktur data linear yang bekerja berdasarkan prinsip FIFO (First In, First Out), yaitu elemen yang pertama kali masuk menjadi elemen yang pertama kali keluar. Konsep ini mirip dengan barisan antrian di kehidupan nyata, seperti antrian di kasir atau loket tiket. Pada queue, proses penambahan elemen hanya dapat dilakukan pada satu sisi yang disebut rear atau tail, sedangkan penghapusan elemen dilakukan dari sisi lain yang disebut front atau head. Secara umum, queue dapat diimplementasikan menggunakan array statis, array dinamis, maupun linked list. Pada praktikum ini, queue diimplementasikan menggunakan array statis berukuran tetap, sehingga proses pengelolaan indeks head dan tail menjadi sangat penting. Terdapat beberapa pendekatan dalam memanfaatkan ruang array, yaitu queue linear dengan shifting, queue linear tanpa shifting, dan circular queue. Pada Queue Linear dengan Shifting, head tetap berada pada posisi awal (biasanya index 0), sementara tail bergerak ke indeks berikutnya setiap kali elemen ditambahkan. Ketika elemen di-dequeue, semua elemen di belakangnya digeser ke kiri untuk mempertahankan posisi head di indeks 0. Pendekatan ini mudah dipahami namun tidak efisien karena memerlukan operasi shifting berkali-kali. Pendekatan lain adalah Queue Linear tanpa Shifting, di mana head dan tail sama-sama bergerak maju mengikuti proses enqueue dan dequeue. Pada model ini, tidak ada proses penggeseran elemen sehingga lebih efisien. Namun, ketika tail mencapai batas akhir array, queue dianggap penuh meskipun masih terdapat ruang kosong di bagian awal array. Hal ini menyebabkan pemanfaatan memori kurang optimal. Untuk mengatasi kelemahan kedua model sebelumnya, digunakan Circular Queue, yaitu mekanisme queue yang memanfaatkan sifat melingkar dari array menggunakan operasi modulo. Ketika tail mencapai indeks terakhir array, ia dapat kembali ke indeks 0 selama posisi tersebut masih kosong. Circular queue memungkinkan seluruh slot array dapat digunakan tanpa adanya pemborosan ruang dan tanpa perlu shifting, sehingga menjadi implementasi paling efisien dari struktur queue berbasis array statis. Queue memiliki berbagai aplikasi praktis dalam bidang ilmu komputer seperti manajemen tugas pada sistem operasi, buffer input/output, penjadwalan proses, pemrosesan antrian data, dan berbagai simulasi antrian nyata. Dengan memahami berbagai cara implementasinya, mahasiswa dapat memilih metode yang sesuai berdasarkan kebutuhan efisiensi dan karakteristik data.

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

Guided 1

```
h queue.h  X
Modul08 > GUIDED > h queue.h > ...
1  #ifndef QUEUE_H // Jika QUEUE_H belum didefinisikan
2  #define QUEUE_H // Maka definisikan QUEUE_H untuk mencegah inklusi ganda
3
4  #define MAX_QUEUE 5 // Menentukan ukuran maksimal antrean
5
6  // Mendefinisikan struktur (tipe data) untuk Queue
7  struct Queue {
8      int info[MAX_QUEUE]; // Array untuk menyimpan data antrean
9      int head; // Penanda untuk elemen paling depan (kepala)
10     int tail; // Penanda untuk elemen paling belakang (ekor)
11     int count; // Penghitung jumlah elemen saat ini
12 };
13
14 // Prosedur untuk membuat queue kosong
15 void createQueue(Queue &Q);
16
17 // Fungsi untuk mengecek apakah queue kosong
18 bool isEmpty(Queue Q);
19
20 // Fungsi untuk mengecek apakah queue penuh
21 bool isFull(Queue Q);
22
23 // Prosedur untuk menambahkan elemen ke queue (enqueue)
24 void enqueue(Queue &Q, int x);
25
26 // Fungsi untuk menghapus dan mengembalikan elemen dari queue (dequeue)
27 int dequeue(Queue &Q);
28
29 // Prosedur untuk menampilkan semua isi queue
30 void printInfo(Queue Q);
31
32 #endif
```

```
C++ queue.cpp X
Modul08 > GUIDED > C++ queue.cpp > printInfo(Queue)

1 #include "queue.h"
2 #include <iostream>
3
4 using namespace std; //Menggunakan namespace standar agar tidak perlu menulis std::
5
6 // Definisi prosedur untuk membuat queue kosong
7 void createQueue(Queue &Q) {
8     Q.head = 0; // Set kepala ke indeks 0
9     Q.tail = 0; // Set ekor ke indeks 0
10    Q.count = 0; // Set jumlah elemen ke 0
11 }
12
13 // Definisi fungsi untuk mengecek apakah queue kosong
14 bool isEmpty(Queue Q) {
15     return Q.count == 0; //Kembalikan true jika jumlah elemen adalah 0
16 }
17
18 // Definisi fungsi untuk mengecek apakah queue penuh
19 bool isFull(Queue Q) {
20     return Q.count == MAX_QUEUE; // Kembalikan true jika jumlah elemen sama dengan maks
21 }
22
23 // Definisi prosedur untuk menambahkan elemen (enqueue)
24 void enqueue(Queue &Q, int x) {
25     if (!isFull(Q)) { // Jika queue tidak penuh
26         Q.info[Q.tail] = x; // Masukkan data (x) ke posisi ekor (tail)
27         // Pindahkan ekor secara circular (memutar)
28         Q.tail = (Q.tail + 1) % MAX_QUEUE;
29         Q.count++; //Tambah jumlah elemen
30     } else { // Jika queue penuh
31         cout << "Antrean Penuh!" << endl; //Tampilkan pesan error
32     }
33 }
34
35 //Definisi fungsi untuk menghapus elemen (dequeue)
36 int dequeue(Queue &Q) {
37     if (!isEmpty(Q)) { // Jika queue tidak kosong
38         int x = Q.info[Q.head]; // Ambil data di posisi kepala (head)
39         //Pindahkan kepala secara circular (memutar)
40         Q.head = (Q.head + 1) % MAX_QUEUE;
41         Q.count--; // Kurangi jumlah elemen
42         return x; // Kembalikan data yang diambil
43     } else { // Jika queue kosong
44         cout << "Antrean Kosong!" << endl; //Tampilkan pesan error
45         return -1; // Kembalikan nilai -1 sebagai tanda error
46     }
47 }
48
49 // Definisi prosedur untuk menampilkan isi queue
50 void printInfo(Queue Q) {
51     cout << "Isi Queue: [ "; // Tampilkan awalan
52     if (!isEmpty(Q)) { // ika tidak kosong
53         int i = Q.head; // Mulai dari kepala
54         int n = 0; //Penghitung elemen yang sudah dicetak
55         while (n < Q.count) { // Ulangi sebanyak jumlah elemen
56             cout << Q.info[i] << " "; // Cetak info
57             i = (i + 1) % MAX_QUEUE; // Geser 'i' secara circular
58             n++; // Tambah penghitung
59         }
60     }
61     cout << "]" << endl; // Tampilkan akhiran
62 }
```

```

C++ main.cpp ×
Modul08 > GUIDED > C++ main.cpp > main()
1 #include <iostream> // Menyertakan library untuk input/output
2 #include "queue.h" // Menyertakan file header queue kita
3 #include "queue.cpp"
4
5 using namespace std; // Menggunakan namespace standar
6
7 // Fungsi utama program
Tabnine | Edit | Test | Explain | Document
8 int main() {
9     Queue Q; // Deklarasikan variabel queue bernama Q
10
11     createQueue(Q); // Panggil prosedur untuk inisialisasi queue
12     printInfo(Q); // Tampilkan isi queue (seharusnya kosong)
13
14     cout << "\n Enqueue 3 Elemen" << endl;
15     enqueue(Q, 5);
16     printInfo(Q);
17     enqueue(Q, 2);
18     printInfo(Q);
19     enqueue(Q, 7);
20     printInfo(Q);
21
22     cout << "\n Dequeue 1 Elemen" << endl;
23     // Hapus 1 elemen dan tampilkan nilainya
24     cout << "Elemen keluar: " << dequeue(Q) << endl;
25     printInfo(Q); // Tampilkan isi queue setelah dequeue
26
27     cout << "\n Enqueue 1 Elemen" << endl;
28     enqueue(Q, 4);
29     printInfo(Q);
30
31     cout << "\n Dequeue 2 Elemen" << endl;
32     // Hapus 1 elemen dan tampilkan nilainya
33     cout << "Elemen keluar: " << dequeue(Q) << endl;
34     // Hapus 1 elemen lagi dan tampilkan nilainya
35     cout << "Elemen keluar: " << dequeue(Q) << endl;
36     printInfo(Q); // Tampilkan isi queue
37
38     return 0;
39 }

```

## Screenshots Output

```

PS D:\StrukturData> cd "d:\StrukturData\Modul08\GUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { ./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 ]

```

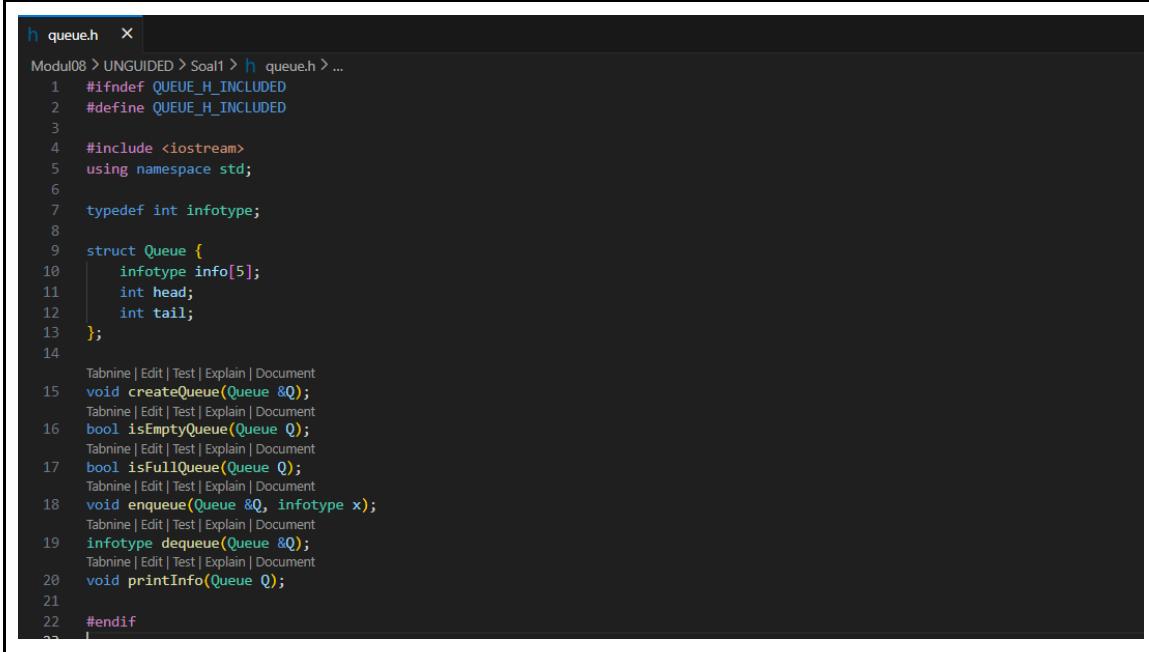
## Deskripsi:

Program ini merupakan implementasi Circular Queue menggunakan array berukuran 5. Queue bekerja dengan prinsip FIFO, di mana elemen ditambahkan melalui enqueue() dan diambil melalui dequeue(). Indeks head dan tail bergerak secara melingkar menggunakan operasi modulo sehingga ruang array dapat digunakan secara efisien. Fungsi isEmpty() dan isFull() mengecek kondisi queue, sedangkan printInfo() menampilkan isinya. Pada main.cpp, beberapa operasi enqueue dan dequeue dilakukan untuk menunjukkan cara

kerja antrian secara circular.

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

#### Unguided 1



The image shows a code editor window with a dark theme. The file being edited is named "queue.h". The code defines a Queue structure with an array of infotype elements, a head pointer, and a tail pointer. It also includes several member functions: createQueue, isEmptyQueue, isFullQueue, enqueue, dequeue, and printInfo. The code is annotated with Tabnine AI-generated documentation comments above each function.

```
#queue.h
Modul08 > UNGUIDED > Soal1 > #queue.h > ...
1  #ifndef QUEUE_H_INCLUDED
2  #define QUEUE_H_INCLUDED
3
4  #include <iostream>
5  using namespace std;
6
7  typedef int infotype;
8
9  struct Queue {
10    infotype info[5];
11    int head;
12    int tail;
13 };
14
15 void createQueue(Queue &Q);
16 bool isEmptyQueue(Queue Q);
17 bool isFullQueue(Queue Q);
18 void enqueue(Queue &Q, infotype x);
19 infotype dequeue(Queue &Q);
20 void printInfo(Queue Q);
21
22 #endif
```

```
C++ queue.cpp X
Modul08 > UNGUIDED > Soal1 > C++ queue.cpp > ...
1 #include "queue.h"
2
3 void createQueue(Queue &Q) {
4     Q.head = -1;
5     Q.tail = -1;
6 }
7
8 bool isEmptyQueue(Queue Q) {
9     return (Q.head == -1 && Q.tail == -1);
10 }
11
12 bool isFullQueue(Queue Q) {
13     return (Q.tail == 4);
14 }
15
16 void enqueue(Queue &Q, infotype x) {
17     if (isFullQueue(Q)) {
18         cout << "Queue penuh!" << endl;
19         return;
20     }
21
22     if (isEmptyQueue(Q)) {
23         Q.head = 0;
24         Q.tail = 0;
25     } else {
26         Q.tail++;
27     }
28
29     Q.info[Q.tail] = x;
30 }
31
32 infotype dequeue(Queue &Q) {
33     if (isEmptyQueue(Q)) {
34         cout << "Queue kosong!" << endl;
35         return -1;
36     }
37
38     infotype x = Q.info[Q.head];
39
40     if (Q.head == Q.tail) {
41         Q.head = -1;
42         Q.tail = -1;
43     } else {
44         for (int i = Q.head; i < Q.tail; i++) {
45             Q.info[i] = Q.info[i + 1];
46         }
47         Q.tail--;
48     }
49
50     return x;
51 }
52
53 void printInfo(Queue Q) {
54     cout << Q.head << " - " << Q.tail << " | ";
55
56     if (isEmptyQueue(Q)) {
57         cout << "empty queue" << endl;
58         return;
59     }
60
61     for (int i = Q.head; i <= Q.tail; i++) {
62         cout << Q.info[i] << " ";
63     }
64     cout << endl;
65 }
66
```

```
C++ main.cpp ×
Modul08 > UNGUIDED > Soal1 > C++ main.cpp > ...
1 #include <iostream>
2 #include "queue.h"
3 #include "queue.cpp"
4
5 using namespace std;
6
7 Tabnine | Edit | Test | Explain | Document
8 int main() {
9     cout << "Hello world!" << endl;
10
11     Queue Q;
12     createQueue(Q);
13
14     cout << "-----" << endl;
15     cout << " H - T \t | Queue info" << endl;
16     cout << "-----" << endl;
17
18     printInfo(Q);
19     enqueue(Q, 5); printInfo(Q);
20     enqueue(Q, 2); printInfo(Q);
21     enqueue(Q, 7); printInfo(Q);
22     dequeue(Q); printInfo(Q);
23     enqueue(Q, 4); printInfo(Q);
24     dequeue(Q); printInfo(Q);
25     dequeue(Q); printInfo(Q);
26
27     return 0;
28 }
```

## Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul08\UNGUIDED\Soal1\" ; if ($?) { g++ main.cpp -o main } ; if (?) { ./main }
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
```

## Deskripsi:

Pada Unguided 1, queue diimplementasikan menggunakan array dengan pendekatan sederhana, yaitu head tetap berada pada posisi indeks 0 sementara tail bergerak ke kanan setiap kali dilakukan operasi enqueue. Ketika elemen dikeluarkan melalui dequeue, elemen pada indeks head diambil dan seluruh elemen lain digeser (shifting) satu langkah ke kiri agar head tetap berada pada indeks 0. Pendekatan ini memudahkan pemahaman konsep dasar FIFO, namun tidak efisien karena setiap operasi dequeue membutuhkan proses penggeseran elemen yang memakan waktu  $O(n)$ . Selain itu, kapasitas array tidak digunakan secara optimal karena hanya tail yang bergerak, sedangkan posisi kosong di depan tidak dapat dimanfaatkan tanpa dilakukan shifting.

## Unguided 2

h queue.h X

Modul08 > UNGUIDED > Soal2 > h queue.h > ...

```
1 #ifndef QUEUE_H_INCLUDED
2 #define QUEUE_H_INCLUDED
3
4 #include <iostream>
5 using namespace std;
6
7 typedef int infotype;
8
9 struct Queue {
10     infotype info[5];
11     int head;
12     int tail;
13 };
14
15 void createQueue(Queue &Q);
16 bool isEmptyQueue(Queue Q);
17 bool isFullQueue(Queue Q);
18 void enqueue(Queue &Q, infotype x);
19 infotype dequeue(Queue &Q);
20 void printInfo(Queue Q);
21
22 #endif
```

```
C++ queue.cpp X
Modul08 > UNGUIDED > Soal2 > C++ queue.cpp > ...
1 #include "queue.h"
2
3 Tabnine | Edit | Test | Explain | Document
4 void createQueue(Queue &Q) {
5     Q.head = -1;
6     Q.tail = -1;
7 }
8
9 Tabnine | Edit | Test | Explain | Document
10 bool isEmptyQueue(Queue Q) {
11     return (Q.head == -1 && Q.tail == -1);
12 }
13
14 Tabnine | Edit | Test | Explain | Document
15 bool isFullQueue(Queue Q) {
16     return (Q.tail == 4);
17 }
18
19 Tabnine | Edit | Test | Explain | Document
20 void enqueue(Queue &Q, infotype x) {
21     if (isFullQueue(Q)) {
22         cout << "Queue penuh!" << endl;
23         return;
24     }
25
26     if (isEmptyQueue(Q)) {
27         Q.head = 0;
28         Q.tail = 0;
29     } else {
30         Q.tail++;
31     }
32
33     Q.info[Q.tail] = x;
34 }
35
36 Tabnine | Edit | Test | Explain | Document
37 infotype dequeue(Queue &Q) {
38     if (isEmptyQueue(Q)) {
39         cout << "Queue kosong!" << endl;
40         return -1;
41     }
42
43     infotype x = Q.info[Q.head];
44
45     if (Q.head == Q.tail) {
46         Q.head = -1;
47         Q.tail = -1;
48     } else {
49         Q.head++;
50     }
51
52     return x;
53 }
54
55 Tabnine | Edit | Test | Explain | Document
56 void printInfo(Queue Q) {
57     cout << Q.head << " - " << Q.tail << " | ";
58
59     if (isEmptyQueue(Q)) {
60         cout << "empty queue" << endl;
61         return;
62     }
63
64     for (int i = Q.head; i <= Q.tail; i++) {
65         cout << Q.info[i] << " ";
66     }
67     cout << endl;
68 }
```

```
C++ main.cpp ×
Modul08 > UNGUIDED > Soal2 > C++ main.cpp > ...
1 #include <iostream>
2 #include "queue.h"
3 #include "queue.cpp"
4
5 using namespace std;
6
7 Tabnine | Edit | Test | Explain | Document
8 int main() {
9     cout << "Hello world!" << endl;
10
11     Queue Q;
12     createQueue(Q);
13
14     cout << "-----" << endl;
15     cout << " H - T \t | queue info" << endl;
16     cout << "-----" << endl;
17
18     printInfo(Q);
19     enqueue(Q, 5); printInfo(Q);
20     enqueue(Q, 2); printInfo(Q);
21     enqueue(Q, 7); printInfo(Q);
22     dequeue(Q); printInfo(Q);
23     enqueue(Q, 4); printInfo(Q);
24     dequeue(Q); printInfo(Q);
25     dequeue(Q); printInfo(Q);
26
27 }
```

## Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul08\UNGUIDED\Soal2\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Hello world!
-----
H - T | Queue info
-----
-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
```

## Deskripsi:

Pada Unguided 2, implementasi queue dibuat lebih efisien dengan menghilangkan proses shifting. Baik head maupun tail dapat bergerak maju seiring berjalannya operasi enqueue dan dequeue. Ketika dequeue dilakukan, head cukup digeser ke kanan tanpa perlu memindahkan elemen lain. Pendekatan ini lebih cepat dibanding Unguided 1 karena tidak ada pergeseran array. Namun, metode ini masih memiliki kekurangan yaitu ruang array dapat terbuang; ketika tail mencapai indeks terakhir (misalnya indeks 4), queue dianggap penuh meskipun sebenarnya masih ada ruang kosong di bagian awal array akibat head yang sudah maju. Hal ini membuat pemanfaatan memori tidak optimal.

```
h queue.h  X
Modul08 > UNGUIDED > Soal3 > h queue.h > ...
1  #ifndef QUEUE_H_INCLUDED
2  #define QUEUE_H_INCLUDED
3
4  #include <iostream>
5  using namespace std;
6
7  typedef int infotype;
8
9  struct Queue {
10    infotype info[5];
11    int head;
12    int tail;
13 };
14
15 Tabnine | Edit | Test | Explain | Document
16 void createQueue(Queue &Q);
17 Tabnine | Edit | Test | Explain | Document
18 bool isEmptyQueue(Queue Q);
19 Tabnine | Edit | Test | Explain | Document
20 bool isFullQueue(Queue Q);
21 Tabnine | Edit | Test | Explain | Document
22 void enqueue(Queue &Q, infotype x);
23 Tabnine | Edit | Test | Explain | Document
24 infotype dequeue(Queue &Q);
25 Tabnine | Edit | Test | Explain | Document
26 void printInfo(Queue Q);
27
28 #endif
```

```
C++ queue.cpp ×
Modul08 > UNGUIDED > Soal3 > C++ queue.cpp > ...
1 #include "queue.h"
2
3 Tabnine | Edit | Test | Explain | Document
4 void createQueue(Queue &Q) {
5     Q.head = -1;
6     Q.tail = -1;
7 }
8
9 Tabnine | Edit | Test | Explain | Document
10 bool isEmptyQueue(Queue Q) {
11     return (Q.head == -1 && Q.tail == -1);
12 }
13
14 Tabnine | Edit | Test | Explain | Document
15 bool isFullQueue(Queue Q) {
16     return ((Q.tail + 1) % 5 == Q.head);
17 }
18
19 Tabnine | Edit | Test | Explain | Document
20 void enqueue(Queue &Q, infotype x) {
21     if (isFullQueue(Q)) {
22         cout << "Queue penuh!" << endl;
23         return;
24     }
25
26     if (isEmptyQueue(Q)) {
27         Q.head = 0;
28         Q.tail = 0;
29     } else {
30         Q.tail = (Q.tail + 1) % 5;
31     }
32
33     Q.info[Q.tail] = x;
34 }
35
36 Tabnine | Edit | Test | Explain | Document
37 infotype dequeue(Queue &Q) {
38     if (isEmptyQueue(Q)) {
39         cout << "Queue kosong!" << endl;
40         return -1;
41     }
42
43     infotype x = Q.info[Q.head];
44
45     if (Q.head == Q.tail) {
46         Q.head = -1;
47         Q.tail = -1;
48     } else {
49         Q.head = (Q.head + 1) % 5;
50     }
51
52     return x;
53 }
54
55 Tabnine | Edit | Test | Explain | Document
56 void printInfo(Queue Q) {
57     if (isEmptyQueue(Q)) {
58         cout << "-1 -1 | empty queue" << endl;
59         return;
60     }
61
62     cout << Q.head << " - " << Q.tail << " | ";
63
64     int i = Q.head;
65     while (true) {
66         cout << Q.info[i] << " ";
67         if (i == Q.tail) break;
68         i = (i + 1) % 5;
69     }
70     cout << endl;
71 }
```

```
C++ main.cpp x
Modul08 > UNGUIDED > Soal3 > C++ main.cpp > ...
1 #include <iostream>
2 #include "queue.h"
3 #include "queue.cpp"
4
5 using namespace std;
6
7 Tabnine | Edit | Test | Explain | Document
8 int main() {
9     cout << "Hello world!" << endl;
10
11     Queue Q;
12     createQueue(Q);
13
14     printInfo(Q);
15     enqueue(Q, 5); printInfo(Q);
16     enqueue(Q, 2); printInfo(Q);
17     enqueue(Q, 7); printInfo(Q);
18     enqueue(Q, 4); printInfo(Q);
19     dequeue(Q); printInfo(Q);
20     enqueue(Q, 9); printInfo(Q);
21     dequeue(Q); printInfo(Q);
22     dequeue(Q); printInfo(Q);
23
24 }
```

## Screenshots Output

```
● PS D:\StrukturData> cd "d:\StrukturData\Modul08\UNGUIDED\Soal3\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Hello world!
-1 -1 | empty queue
0 -0 | 5
0 -1 | 5 2
0 -2 | 5 2 7
0 -3 | 5 2 7 4
1 -3 | 2 7 4
1 -4 | 2 7 4 9
2 -4 | 7 4 9
3 -4 | 4 9
```

### Deskripsi:

Unguided 3 menggunakan mekanisme circular queue, yaitu bentuk queue yang memanfaatkan array secara melingkar menggunakan operasi modulo. Baik head maupun tail dapat kembali ke indeks awal ketika mencapai batas akhir array, sehingga seluruh slot array dapat digunakan tanpa ada yang terbuang. Queue dianggap penuh ketika  $(tail + 1) \% size == head$ , sementara queue kosong ketika head dan tail bernilai -1. Dengan pendekatan ini, tidak ada shifting, tidak ada ruang yang terbuang, dan operasi enqueue maupun dequeue tetap efisien meskipun array memiliki kapasitas terbatas. Ini adalah implementasi queue paling optimal dan menyerupai penggunaan queue pada sistem nyata.

## D. Kesimpulan

Praktikum ini menunjukkan bahwa struktr data Queue dapat diimplementasikan dengan berbagai cara, dan penggunaan Circular Queue adalah metode yang paling efisien ketika menggunakan array statis. Dengan memanfaatkan pergerakan head dan tail secara melingkar, seluruh kapasitas array dapat digunakan tanpa perlu melakukan shifting elemen, sehingga operasi enqueue dan dequeue menjadi lebih optimal. Melalui percobaan

yang dilakukan, mahasiswa dapat memahami prinsip kerja FIFO serta perbedaan efisiensi antara implementasi queue linear dan circular queue.

#### E. Referensi

Ananda, D. (2020). Struktur Data Queue: Pengertian, Fungsi, dan Jenis-Jenisnya. Dicoding Indonesia.

<https://www.dicoding.com/blog/struktur-data-queue-pengertian-fungsi-dan-jenisnya/>

Softwareseni. (2021). Queue: Pengertian, Tipe, dan Contoh Implementasi.

<https://www.softwareseni.co.id/blog/queue-adalah-pengertian-tipe-dan-contoh-implementasi>

Nblognlife. (2014). Konsep Queue dalam C++.

<https://www.nblognlife.com/2014/05/c-konsep-queue.html>