

## JOB SHEET VIII

### QUEUE

#### 8.1 Tujuan Praktikum

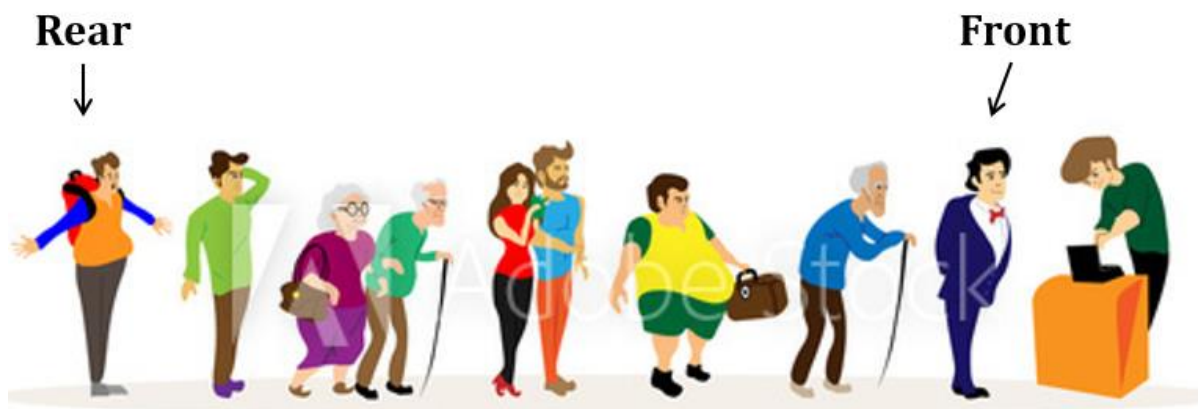
Setelah melakukan materi praktikum ini, mahasiswa mampu:

1. Menjelaskan mengenai algoritma Queue
2. Membuat dan mendeklarasikan struktur algoritma Queue
3. Menerapkan dan mengimplementasikan algoritma Queue

#### 8.2 Ulasan Teori

**Queue**, sering disebut dengan istilah antrian, merupakan sekumpulan elemen/data dimana proses **memasukkan/menambah** elemen/data dilakukan pada posisi **belakang** (*rear*) dan proses **mengeluarkan/mengambil** elemen/data dilakukan pada elemen/data di posisi **depan** (*front*). Istilah untuk menggambarkan kondisi tersebut adalah **FIFO (First In First Out)**. Jadi proses penambahan data hanya bisa dilakukan di posisi paling belakang, dan sebaliknya, proses penghapusan/pengambilan data dilakukan di posisi paling depan.

Queue dapat diilustrasikan seperti barisan orang yang sedang mengantri membeli tiket, orang yang pertama datang ke lokasi akan dilayani terlebih dahulu. Contoh lainnya adalah pasien yang mengantri untuk diperiksa oleh dokter. Ilustrasi mengenai Queue dapat dilihat pada gambar berikut.

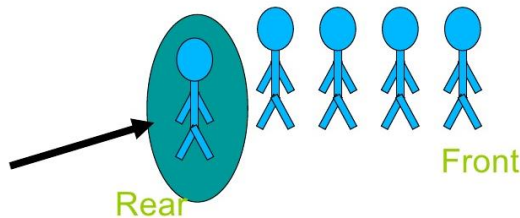


Queue mempunyai dua elemen utama yaitu elemen pertama yang disebut Head atau Front dan elemen terakhir yang disebut Tail atau Rear. Penambahan elemen selalu dilakukan setelah elemen terakhir. Sedangkan penghapusan elemen selalu dilakukan pada elemen pertama.

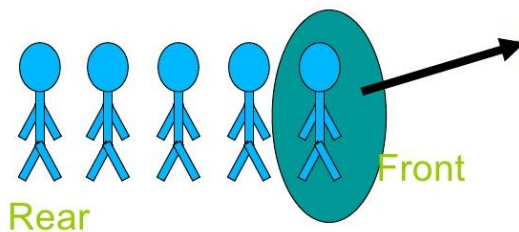
### 8.2.1 Operasi Queue

Queue mempunyai beberapa operasi yang diterapkan, antara lain sebagai berikut:

1. **Create**: proses awal pembuatan queue
2. **IsFull**: mengecek apakah queue dalam kondisi penuh
3. **IsEmpty**: mengecek apakah queue dalam kondisi kosong
4. **Enqueue**: menambah data dalam queue pada posisi paling belakang



5. **Dequeue**: mengambil data dari queue pada posisi paling depan



6. **Peek**: mengecek data paling depan
7. **Print**: menampilkan semua data pada queue

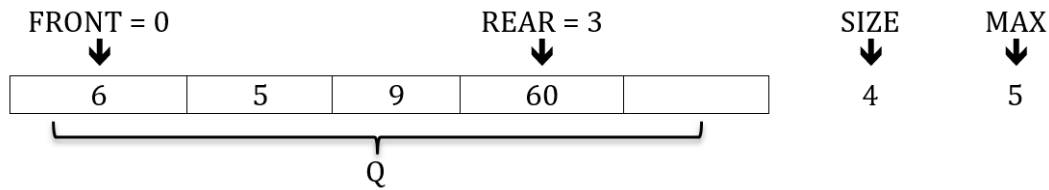
### 8.2.2 Implementasi Queue

Dalam melakukan implementasi queue, terdapat dua cara yang sering digunakan yaitu implementasi menggunakan array dan implementasi menggunakan linked list.

- a. Implementasi menggunakan array
  - Panjang queue bersifat statis
  - Jika queue dibuat dengan panjang 5, maka maksimal data yang bisa ditampung oleh queue tersebut adalah sebanyak 5
- b. Implementasi menggunakan linked list
  - Panjang queue bersifat dinamis
  - Jumlah data yang bisa dimasukkan ke dalam queue bisa bertambah sesuai dengan panjang yang diinginkan

Misalkan terdapat antrian  $Q$  dengan elemen/data sebanyak  $N$  di dalamnya ( $Q_1, Q_2, \dots, Q_N$ ). Data di posisi paling depan antrian  $Q$  disimbolkan  $FRONT(Q)$  dan bagian paling belakang sebagai  $REAR(Q)$ . Jumlah elemen di dalam queue dinyatakan dengan simbol  $SIZE(Q)$  yang dapat dihitung dengan cara

$REAR - FRONT + 1$ . Jadi untuk antrian  $Q = [Q_1, Q_2, \dots, Q_N]$ , maka  $FRONT(Q) = Q_1$ ,  $REAR(Q) = Q_N$  dan  $SIZE(Q) = N$ . Berikut ini merupakan ilustrasi konsep queue yang diimplementasikan menggunakan array.



Penjelasan elemen-elemen yang terdapat pada gambar tersebut adalah sebagai berikut:

- **FRONT** : variabel untuk menyimpan nilai indeks array data terdepan
- **REAR** : variabel untuk menyimpan nilai indeks array data paling belakang
- **SIZE** : variabel untuk menyimpan berapa banyak data yang ada dalam antrian
- **MAX** : variabel untuk menyimpan banyak data maksimal yang bisa disimpan di dalam queue
- **Q** : variabel untuk menyimpan data queue

Dua operasi utama yang sering digunakan pada Queue adalah Enqueue dan Dequeue.

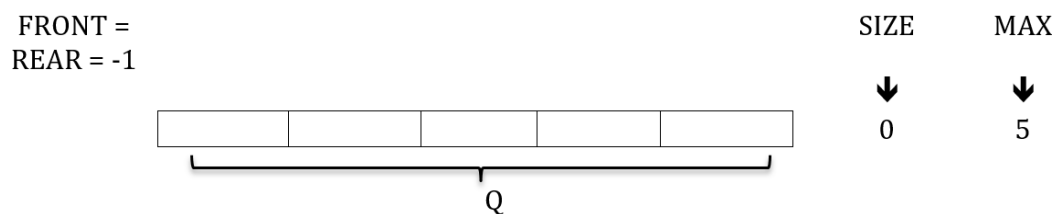
#### OPERASI ENQUEUE

Operasi Enqueue merupakan proses penambahan data baru ke dalam queue. Pada proses enqueue, data baru akan menempati pada posisi paling akhir dalam queue.

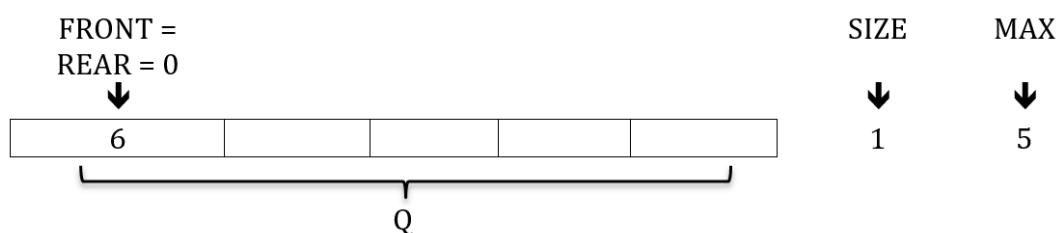
Terdapat 3 kemungkinan kondisi yang terjadi saat Enqueue:

##### 1. Ketika queue dalam kondisi kosong

Ketika terdapat queue dalam kondisi kosong seperti diilustrasikan dalam gambar berikut:

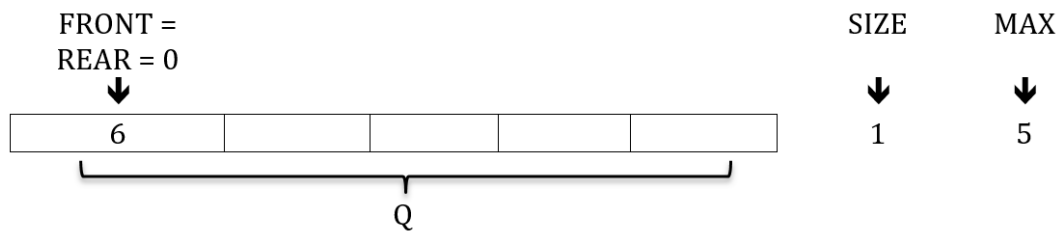


Kemudian hendak dilakukan proses penambahan data, maka data baru dimasukkan ke dalam queue pada indeks 0. Dan data tersebut akan menjadi data pada posisi FRONT dan sekaligus pada posisi REAR. Kemudian SIZE akan bertambah 1. Misalkan data 6 dimasukkan ke dalam queue tersebut, maka bisa diilustrasikan sebagai berikut:

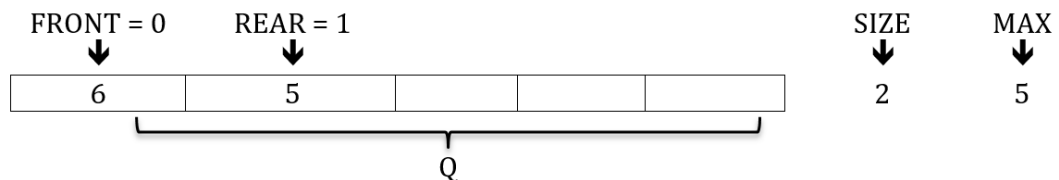


## 2. Ketika data paling belakang dari queue tidak berada di indeks terakhir array

Ketika terdapat queue dimana data paling belakang dari queue tidak berada di indeks terakhir array, seperti ditunjukkan pada gambar berikut:

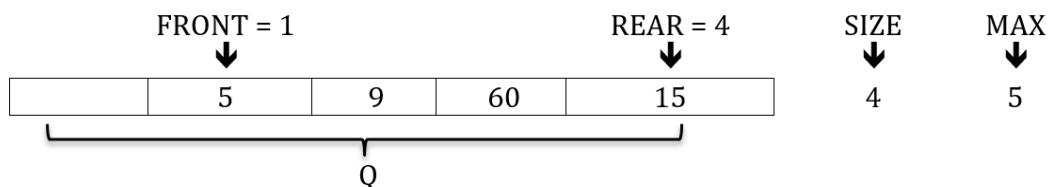


Kemudian akan dimasukkan data baru ke dalam queue tersebut, maka data baru tersebut akan menempati posisi setelah data paling belakang saat ini. Artinya data tersebut akan menempati indeks REAR+1, dan SIZE akan bertambah 1. Misal akan dimasukkan data baru 5 ke dalam queue, maka bisa diilustrasikan sebagai berikut:

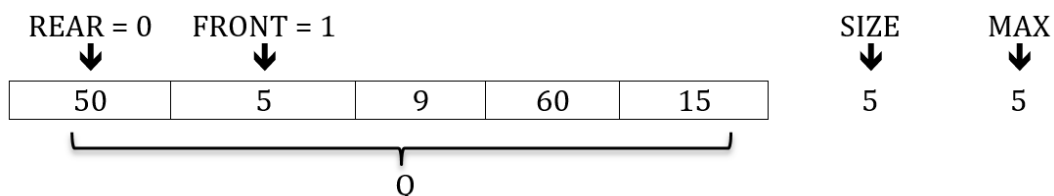


## 3. Ketika data paling belakang dari queue berada di indeks terakhir array

Ketika terdapat queue dimana data paling belakang dari queue berada di indeks terakhir array, seperti ditunjukkan pada gambar berikut:



Kemudian akan dimasukkan data baru ke dalam queue tersebut, maka data baru tersebut akan menempati posisi indeks 0. Artinya posisi REAR selanjutnya adalah 0, dan SIZE akan bertambah 1. Misal akan dimasukkan data baru 50 ke dalam queue, maka bisa diilustrasikan sebagai berikut:



Secara umum, operasi Enqueue dapat dituliskan dalam algoritma sebagai berikut:

1. Memastikan bahwa queue tidak dalam kondisi penuh. Jika queue penuh, maka data tidak bisa dimasukkan ke dalamnya.

2. Jika tidak penuh, maka proses penambahan data ke dalam queue bisa dilakukan.
  - a. Terlebih dulu cek apakah queue dalam kondisi kosong (belum ada data sama sekali).  
Jika ternyata queue masih kosong, berarti data yang akan masuk menjadi data yang paling depan dan sekaligus menjadi data yang paling akhir dalam queue, yaitu pada posisi indeks 0. Artinya  $FRONT = REAR = 0$
  - b. Jika queue dalam kondisi tidak kosong (sudah ada data sebelumnya di dalam queue), yang selanjutnya dilakukan adalah:
    - i. Cek apakah posisi REAR berada pada indeks terakhir array. Jika benar, maka posisi REAR selanjutnya adalah di indeks 0
    - ii. Jika posisi REAR tidak berada pada indeks terakhir array, maka posisi REAR selanjutnya adalah  $REAR + 1$
  - c. Masukkan data ke dalam queue pada indeks REAR
  - d. SIZE bertambah 1

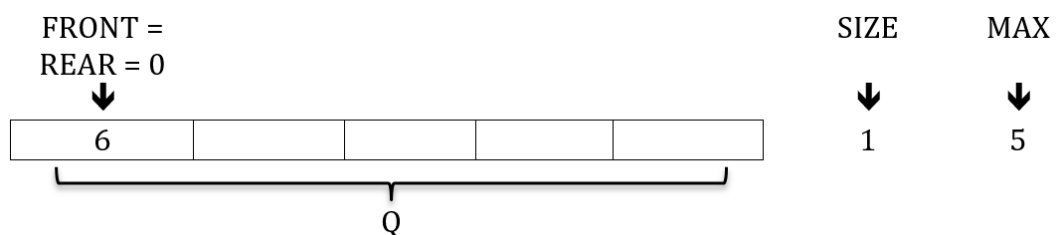
## OPERASI DEQUEUE

Operasi Dequeue merupakan proses pengambilan data dari dalam queue. Pada proses dequeue, data yang akan diambil adalah data yang menempati pada posisi paling depan (FRONT).

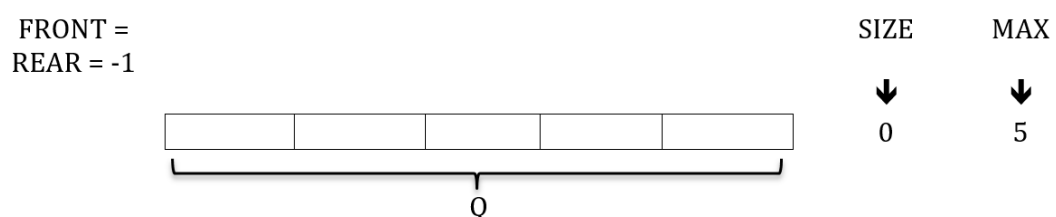
Terdapat 3 kemungkinan kondisi yang terjadi saat Dequeue:

### 1. Ketika queue dalam kondisi kosong setelah data diambil

Ketika terdapat suatu queue, yang di dalamnya hanya tertinggal 1 data saja, seperti diilustrasikan pada gambar berikut:

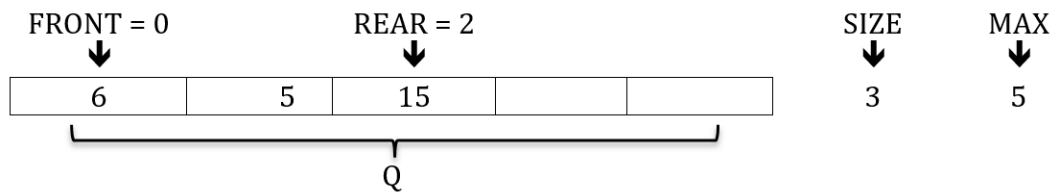


Kemudian hendak diambil data, maka data yang diambil adalah 6. Dan selanjutnya SIZE akan menjadi 0 dan posisi FRONT serta REAR akan diset menjadi -1. Selanjutnya queue berada pada kondisi kosong.

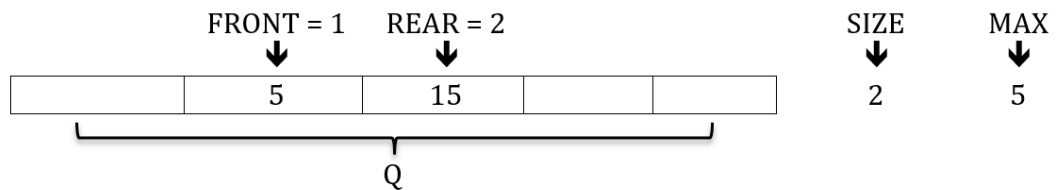


## 2. Ketika data yang paling depan dari queue tidak berada di indeks terakhir array

Ketika terdapat suatu queue dimana data paling depan dari queue tidak berada di indeks terakhir array, seperti ditunjukkan pada gambar berikut:

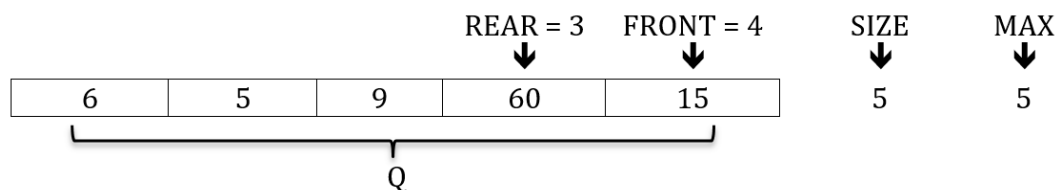


Ketika dilakukan pengambilan data dari queue tersebut, maka data yang terambil adalah 6. Selanjutnya SIZE berkurang 1, dan posisi FRONT akan bertambah 1 dari posisi FRONT sebelumnya.

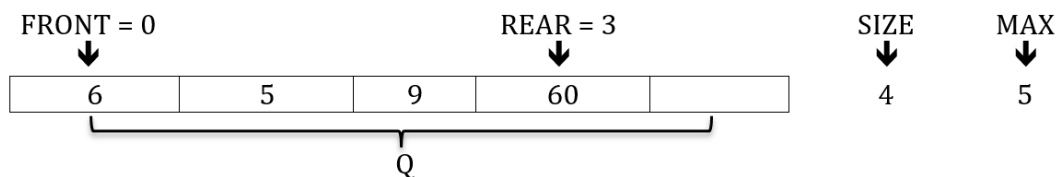


## 3. Ketika data paling depan dari queue berada di indeks terakhir array

Ketika terdapat suatu queue dimana data paling depan dari queue berada di indeks terakhir array, seperti ditunjukkan pada gambar berikut:



Ketika dilakukan pengambilan data dari queue tersebut, maka data yang terambil adalah 15. Selanjutnya SIZE berkurang 1, dan posisi FRONT akan bergeser ke indeks 0 seperti ilustrasi pada gambar berikut:



Secara umum, operasi Dequeue dapat dituliskan dalam algoritma sebagai berikut:

1. Memastikan bahwa queue tidak dalam kondisi kosong. Jika queue kosong, maka tidak ada data yang bisa diambil.
2. Jika tidak kosong, maka proses pengambilan data dari queue bisa dilakukan.
  - a. Ambil data yang ada di indeks FRONT, dimana data tersebut akan di return-kan dari proses ini

- b. SIZE berkurang 1
- c. Selanjutnya, ubah posisi FRONT:
  - i. Cek apakah setelah diambil datanya, queue dalam kondisi kosong ( $SIZE = 0$ ). Jika benar, maka posisi  $FRONT = REAR = -1$
  - ii. Jika setelah diambil datanya dan queue tidak kosong, yang selanjutnya dilakukan adalah:
    - 1. Cek apakah posisi FRONT saat ini berada di indeks terakhir array. Jika benar, maka FRONT selanjutnya diletakkan di indeks 0
    - 2. Jika posisi FRONT tidak berada di indeks terakhir array, maka posisi FRONT selanjutnya adalah FRONT sebelumnya ditambah 1

Dengan demikian, dapat disimpulkan bahwa perubahan indeks front bertambah 1 setiap kali terjadi operasi Dequeue dan indeks rear bertambah 1 setiap kali terjadi operasi Enqueue. Untuk gambaran lebih jelas, perhatikan gambar berikut.

Kondisi sekarang:



Kondisi setelah enqueue:



Kondisi setelah dequeue:

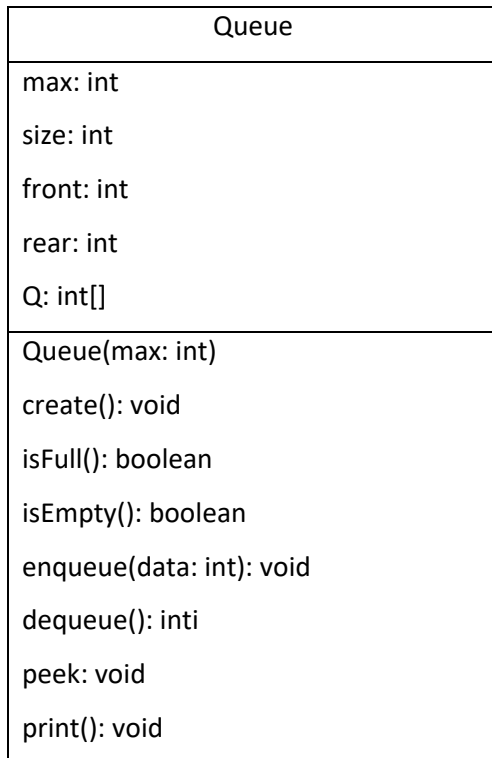


## 8.3 Praktikum

Pada percobaan ini, kita akan mengimplementasikan penggunaan class Queue.

### 8.3.1 Langkah-langkah Percobaan

1. Perhatikan Diagram Class berikut ini:



Berdasarkan diagram class tersebut, akan dibuat program class Queue dalam Java.

2. Pada Project **StrukturData** yang sudah dibuat pada Minggu sebelumnya, buat package dengan nama **minggu8**, kemudian buat class baru dengan nama **Queue**.
3. Tambahkan atribut max, size, front, rear, dan Q sesuai dengan diagram class di atas.
4. Tambahkan pula konstruktor berparameter dan method **Create** seperti gambar berikut ini.

```
public Queue(int n) {  
    max = n;  
    Create();  
}
```

Di dalam konstruktor, terdapat kode program yang digunakan untuk memanggil method **Create**.

```
public void Create() {  
    Q = new int[max];  
    size = 0;  
    front = rear = -1;  
}
```



5. Buat method **IsEmpty** bertipe boolean yang digunakan untuk mengecek apakah queue kosong.

```
public boolean IsEmpty() {  
    if (size == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

6. Buat method **IsFull** bertipe boolean yang digunakan untuk mengecek apakah queue sudah penuh.

```
public boolean IsFull() {  
    if (size == max) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

7. Buat method **peek** bertipe void untuk memeriksa elemen queue pada posisi paling depan.

```
public void peek() {  
    if (!IsEmpty()) {  
        System.out.println("Elemen terdepan: " + Q[front]);  
    } else {  
        System.out.println("Antrian masih kosong");  
    }  
}
```

8. Buat method **print** bertipe void untuk menampilkan seluruh elemen pada queue mulai dari posisi front sampai dengan posisi rear.

```

public void print() {
    if (IsEmpty()) {
        System.out.println("Antrian masih kosong");
    } else {
        int i = front;
        while (i != rear) {
            System.out.print(Q[i] + " ");
            i = (i + 1) % max;
        }
        System.out.println(Q[i] + " ");
        System.out.println("Jumlah antrian = " + size);
    }
}

```

9. Buat method **Enqueue** bertipe void untuk menambahkan isi queue dengan parameter **data** yang bertipe integer

```

public void Enqueue(int data) {
    if (IsFull()) {
        System.out.println("Antrian sudah penuh");
    } else {
        if (IsEmpty()) {
            front = rear = 0;
        } else {
            if (rear == max - 1) {
                rear = 0;
            } else {
                rear++;
            }
        }
        Q[rear] = data;
        size++;
    }
}

```

10. Buat method **Dequeue** bertipe void untuk mengeluarkan data pada queue di posisi belakang

```
public int Dequeue() {
    int data = 0;
    if (IsEmpty()) {
        System.out.println("Antrian masih kosong");
    } else {
        data = Q[front];
        size--;
        if (IsEmpty()) {
            front = rear = -1;
        } else {
            if (front == max - 1) {
                front = 0;
            } else {
                front++;
            }
        }
    }
    return data;
}
```

11. Buat class baru dengan nama **QueueMain** tetap pada package **minggu8**. Buat method menu bertipe void untuk memilih menu program pada saat dijalankan.

```
public static void menu() {
    System.out.println("Masukkan operasi yang diinginkan:");
    System.out.println("1. Enqueue");
    System.out.println("2. Dequeue");
    System.out.println("3. Print");
    System.out.println("4. Peek");
    System.out.println("-----");
}
```

12. Buat method main, kemudian deklarasikan Scanner dengan nama **sc**.  
13. Buat variabel **n** untuk menampung masukan jumlah maksimal data pada queue yang dimasukkan oleh pengguna.

```
System.out.print("Masukkan jumlah maksimal antrian: ");
int n = sc.nextInt();
```

14. Lakukan instansiasi objek dengan nama **Q**.

```
Queue Q = new Queue(n);
```

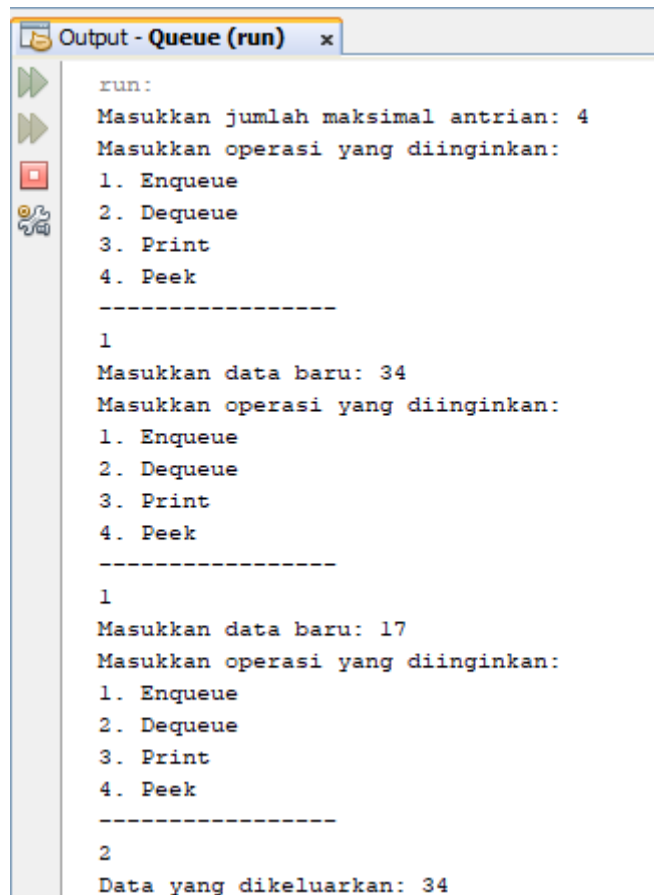
15. Deklarasikan variabel dengan nama **pilih** untuk menampung pilih menu dari pengguna.
16. Lakukan perulangan menggunakan do-while untuk menjalankan program secara terus menerus sesuai masukan yang diberikan. Di dalam perulangan tersebut, terdapat pemilihan kondisi menggunakan switch-case untuk menjalankan operasi queue sesuai dengan masukan pengguna.

```
do {  
    menu();  
    pilih = sc.nextInt();  
    switch (pilih) {  
        case 1:  
            System.out.print("Masukkan data baru: ");  
            int dataMasuk = sc.nextInt();  
            Q.Enqueue(dataMasuk);  
            break;  
        case 2:  
            int dataKeluar = Q.Dequeue();  
            if (dataKeluar != 0) {  
                System.out.println("Data yang dikeluarkan: " + dataKeluar);  
            }  
            break;  
        case 3:  
            Q.print();  
            break;  
        case 4:  
            Q.peek();  
            break;  
    }  
} while (pilih == 1 || pilih == 2 || pilih == 3 || pilih == 4);
```

17. Jalankan (run) class **QueueMain** dan amati hasilnya.

### 8.3.2 Verifikasi Hasil Percobaan

Samakan hasil compile kode program Anda dengan gambar berikut ini.



```
run:
Masukkan jumlah maksimal antrian: 4
Masukkan operasi yang diinginkan:
1. Enqueue
2. Dequeue
3. Print
4. Peek
-----
1
Masukkan data baru: 34
Masukkan operasi yang diinginkan:
1. Enqueue
2. Dequeue
3. Print
4. Peek
-----
1
Masukkan data baru: 17
Masukkan operasi yang diinginkan:
1. Enqueue
2. Dequeue
3. Print
4. Peek
-----
2
Data yang dikeluarkan: 34
```

### 8.3.3 Pertanyaan

1. Perhatikan class **Queue**, apakah fungsi dari atribut **Q**?
2. Pada method **Create**, mengapa atribut **front** dan **rear** diinisialisasi dengan nilai **-1**, tidak **0**? Jelaskan!
3. Perhatikan method **IsFull**, jika kondisi **IF** diubah menjadi **size == max-1**, apa yang terjadi? Mengapa demikian?
4. Pada method **Enqueue**, jelaskan maksud dan kegunaan dari potongan kode berikut!

```
if (rear == max - 1) {
    rear = 0;
```

5. Perhatikan kembali method **Enqueue**, baris kode program manakah yang menunjukkan bahwa data baru disimpan pada posisi terakhir di dalam queue?
6. Mengapa method **Dequeue** mempunyai tipe kembalian **int**, tidak bertipe **void**?
7. Perhatikan kembali method **Dequeue**, baris kode program manakah yang menunjukkan bahwa data yang dikeluarkan adalah data pada posisi paling depan di dalam queue?

8. Pada method **Dequeue**, jelaskan maksud dan kegunaan dari potongan kode berikut!

```
if (front == max - 1) {  
    front = 0;  
}
```

9. Pada method **print**, mengapa pada proses perulangan variabel *i* tidak dimulai dari 0 (**int i=0**), melainkan **int i=front**?
10. Perhatikan kembali method **print**, jelaskan maksud dari potongan kode berikut!

```
i = (i + 1) % max;
```

11. Lakukan modifikasi program dengan menambahkan method baru bernama **peekRear** yang digunakan memeriksa data yang berada di posisi belakang! Tambahkan pula daftar menu class **QueueMain** sehingga method **peekRear** dapat dipanggil!

#### 8.4 Tugas

1. Tambahkan dua method berikut ke dalam class **Queue** pada Praktikum:
- Method peekPosition(data: int) : void** → menampilkan posisi antrian dari sebuah data
  - Method peekAt(position: int) : void** → menampilkan data yang berada pada antrian tertentu

Sesuaikan daftar menu yang terdapat pada class **QueueMain** sehingga kedua method tersebut dapat dipanggil!

2. Buat program antrian nasabah di suatu bank. Ketika seorang nasabah akan antri, maka ia harus menuliskan terlebih dulu no. rekening, dan nama. Jadi, antrian yang akan dibuat, berisi data-data nasabah berupa no. rekening dan nama. Sehingga pertama kali yang harus dibuat adalah class **Nasabah** sebagai berikut:

Nasabah
noRekening: String nama: String
Nasabah(noRek: String, nm: String) Nasabah() print(): void

Selanjutnya buatlah class Queue sebagai berikut:

Queue
max: int front: int rear: int size: int q: Nasabah[]
Queue(max: int) create(): void isEmpty(): boolean isFull(): boolean enqueue(data: int): void dequeue(): int print(): void peek(): void peekRear(): void peekPosition(nas: Nasabah): void printNasabah(posisi: int): void

Catatan:

- Method create(), isEmpty(), isFull(), enqueue(), dequeue() dan print(), kegunaannya sama seperti yang telah dibuat pada Praktikum
- Method peek(): digunakan untuk menampilkan data Nasabah yang ada di posisi antrian paling depan
- Method peekRear(): digunakan untuk menampilkan data Nasabah yang ada di posisi antrian paling belakang
- Method peekPosition(): digunakan untuk menampilkan posisi antrian ke berapa, seorang Nasabah berada
- Method printNasabah(): digunakan untuk menampilkan data nasabah pada suatu posisi tertentu dalam antrian