



# Pertemuan 10

## Queue

Tim Ajar Struktur Data 2021

# Definisi Queue

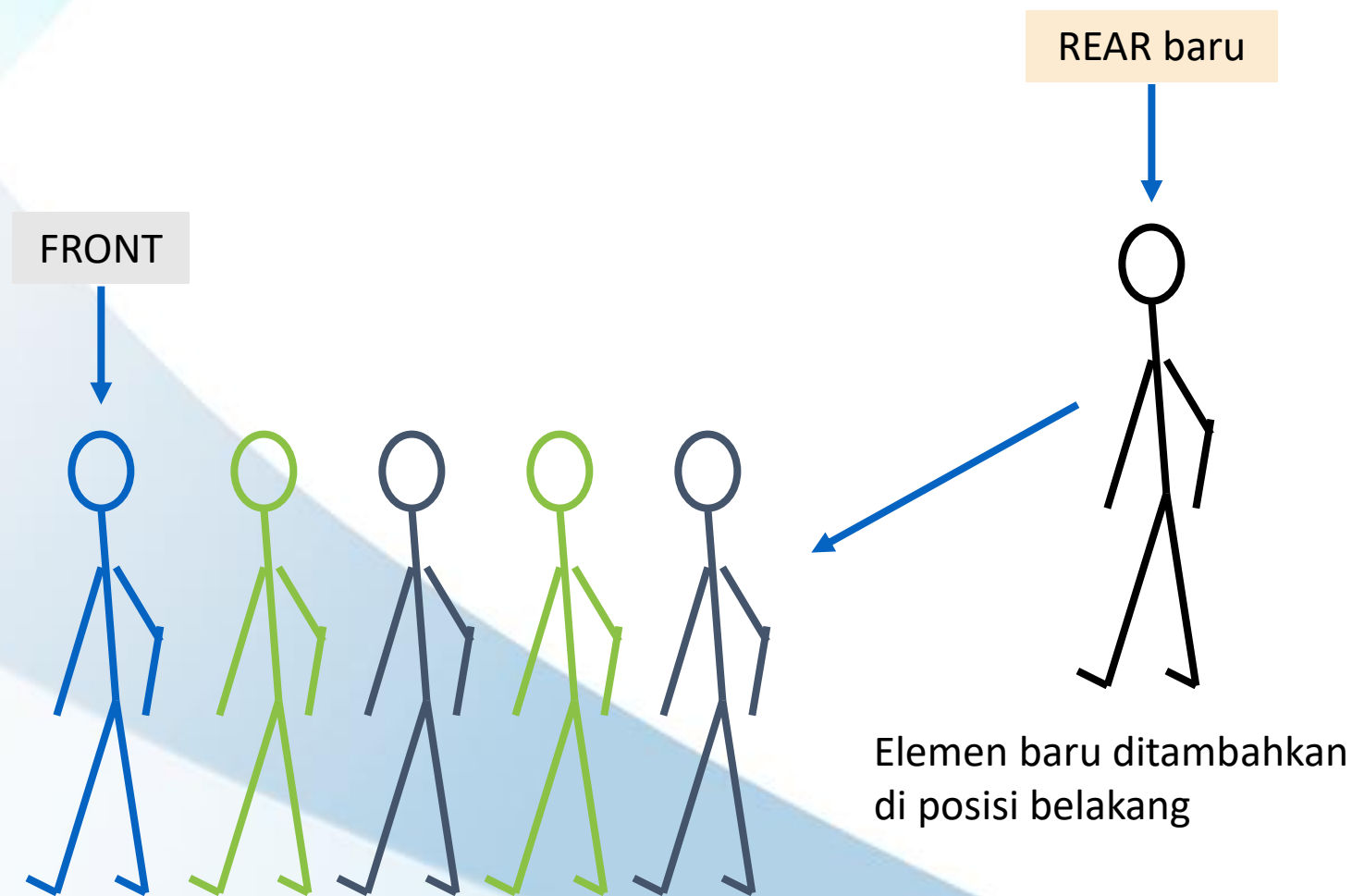
- Queue merupakan struktur data linier yang menerapkan prinsip **First In First Out (FIFO)**
- Proses **menambah** elemen dilakukan pada posisi **belakang** (*rear*) dan proses **mengambil** elemen dilakukan pada elemen di posisi **depan** (*front*)
- Queue disebut juga **antrian**
- Ilustrasi:
  - Barisan orang yang mengantri untuk membeli tiket, orang yang pertama datang akan dilayani terlebih dahulu
  - Antrian job di dalam sistem operasi

# Konsep Queue

- Queue mempunyai dua elemen, yaitu
  - Elemen pertama yang disebut Head / Front
  - Elemen terakhir yang disebut Tail / Rear
- Penambahan elemen selalu dilakukan setelah elemen terakhir
- Penghapusan elemen selalu dilakukan pada elemen pertama

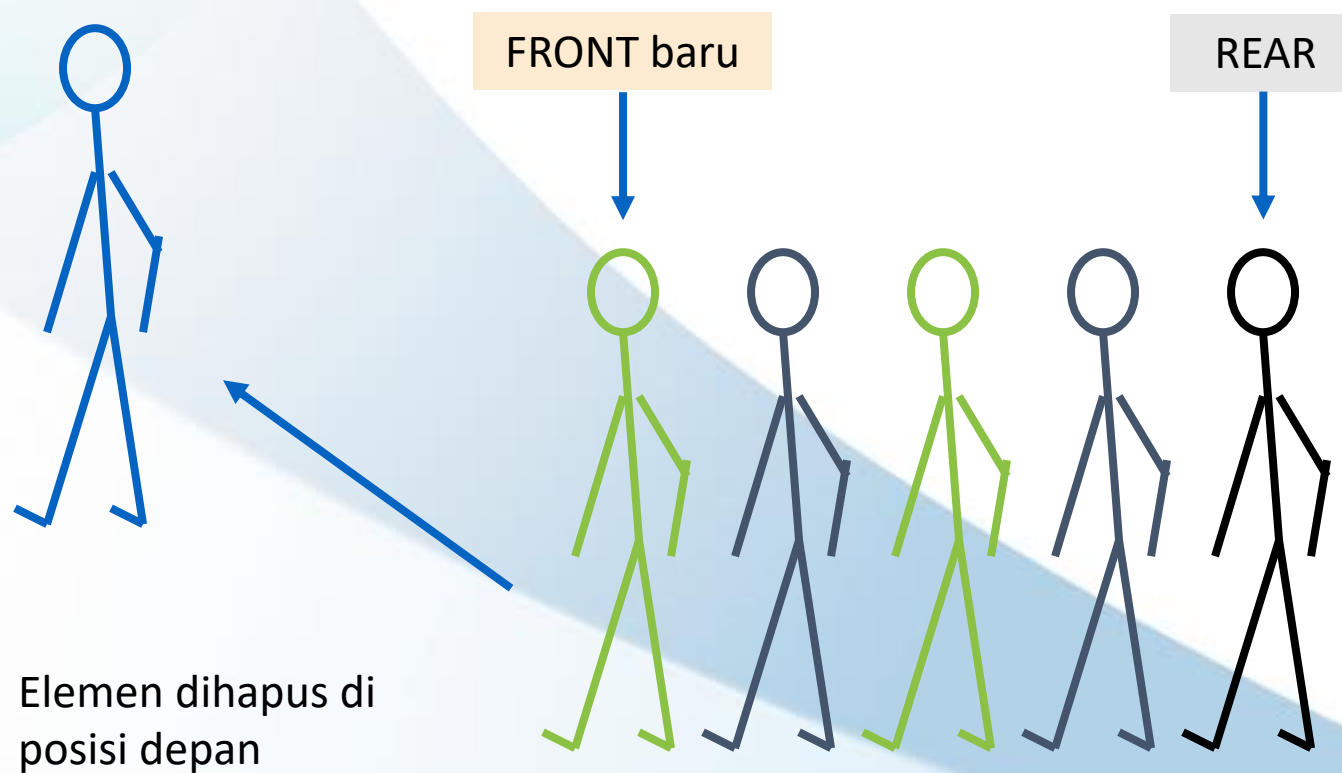
# Konsep Queue

- Menambah elemen



# Konsep Queue

- Menghapus elemen

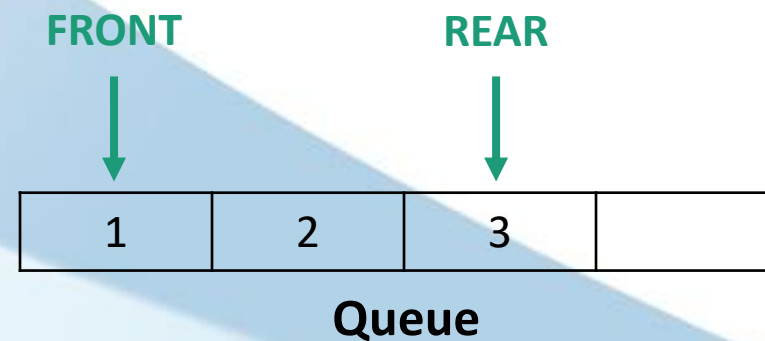
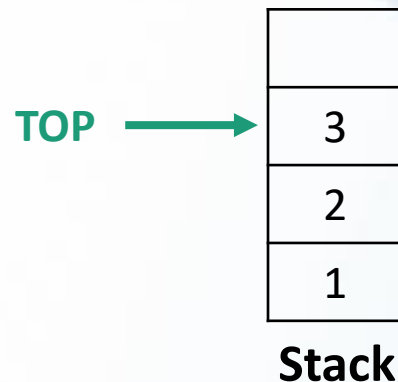


# Operasi Queue

- **IsFull**: mengecek apakah queue dalam kondisi penuh
- **IsEmpty**: mengecek apakah queue dalam kondisi kosong
- **Enqueue**: menambah data dalam queue pada posisi paling belakang
- **Dequeue**: mengambil data dari queue pada posisi paling depan
- **Peek**: mengecek data paling depan
- **Print**: menampilkan semua data pada queue
- **Clear**: menghapus semua elemen yang terdapat pada Queue

# Implementasi Queue

- Implementasi Queue lebih sulit daripada Stack
- Pada Stack, penambahan dan penghapusan data hanya dilakukan pada salah satu sisi saja, sehingga hanya perlu mengubah posisi pointer (TOP) sesuai dengan penambahan atau pengurangan data
- Pada Queue, pengubahan posisi dilakukan pada dua buah pointer, yaitu FRONT dan REAR



# Implementasi Queue

- Menggunakan **Array**:
  - Panjang queue bersifat **statis**
  - Jika dibuat queue dengan panjang 5, maka maksimal queue tersebut bisa menampung 5 data
- Menggunakan **Linked List**:
  - Panjang queue bersifat **dinamis**
  - Jumlah data yang bisa dimasukkan ke dalam queue bisa bertambah sesuai dengan yang diinginkan
- Pembahasan mengenai Linked List tidak akan disampaikan pada pertemuan ini karena akan dibahas pada pertemuan berikutnya



# Implementasi Queue

Misalkan terdapat queue  $Q$  dengan elemen sebanyak  $N$  ( $Q_1, Q_2, \dots, Q_N$ )

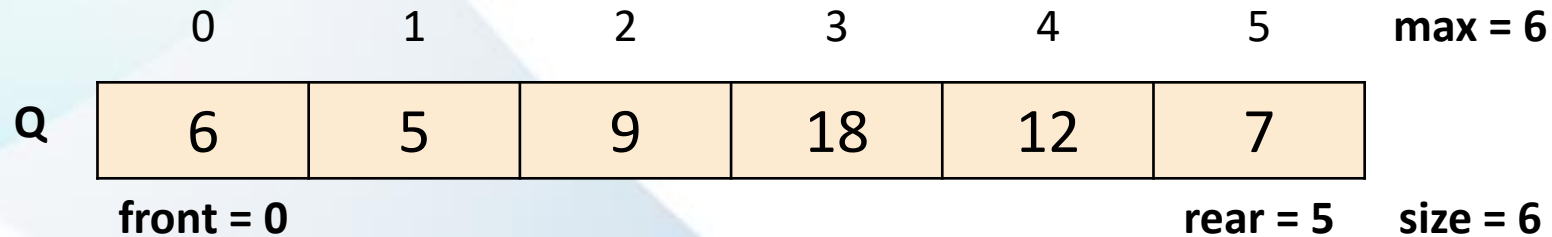
- Data di posisi **depan** queue  $Q$  disimbolkan **FRONT(Q)**
- Data di posisi **belakang** queue  $Q$  disimbolkan **REAR(Q)**
- **Jumlah elemen** di dalam queue dinyatakan dengan simbol **SIZE(Q)** yang dapat dihitung dengan cara **rear – front + 1**
- Untuk queue  $Q = [Q_1, Q_2, \dots, Q_N]$ , maka  
FRONT(Q) =  $Q_1$   
REAR(Q) =  $Q_N$   
SIZE(Q) =  $N$

# Implementasi Queue dengan Array

- **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 array untuk menyimpan data queue

# Implementasi Queue dengan Array

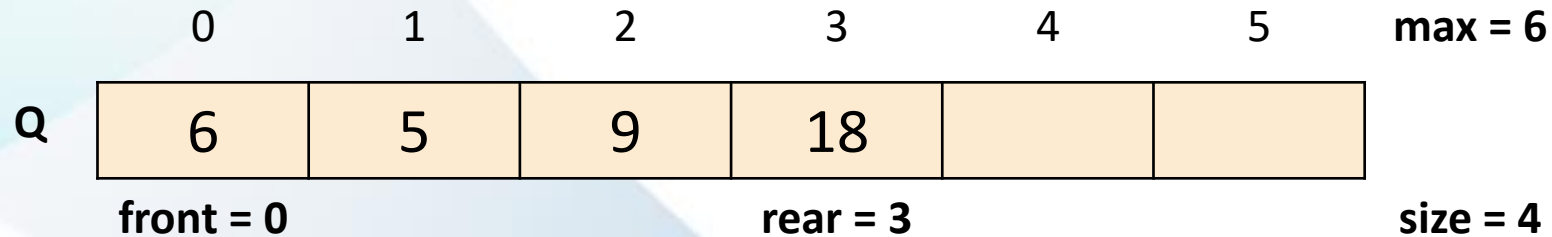
- Ilustrasi ketika queue sudah penuh



- Queue sudah terisi penuh dan tidak dapat menerima data lagi

# Implementasi Queue dengan Array

- Ilustrasi ketika queue belum penuh



- Queue belum penuh sehingga masih dapat menerima data lagi

# Deklarasi Queue

- Proses pertama yang dilakukan adalah deklarasi atau menyiapkan tempat untuk queue
- Langkah-langkah:
  - Deklarasi class
  - Deklarasi front dan rear
  - Deklarasi size dan max
  - Deklarasi array

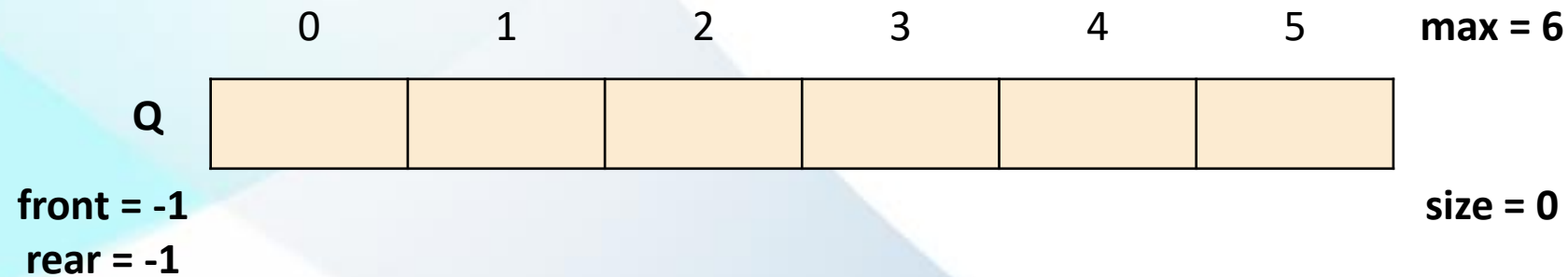
```
public class Queue {  
    int front;  
    int rear;  
    int size;  
    int max;  
    int[] Q;  
}
```

# Inisialisasi (Create) Queue

- Pada awal pembuatan queue, variabel yang perlu diinisialisasi adalah **size** bernilai 0 karena array masih kosong
- Selain itu, **front** dan **rear** bernilai -1 karena tidak menunjuk ke data manapun

# Inisialisasi (Create) Queue

- Ilustrasi queue pada saat inisialisasi



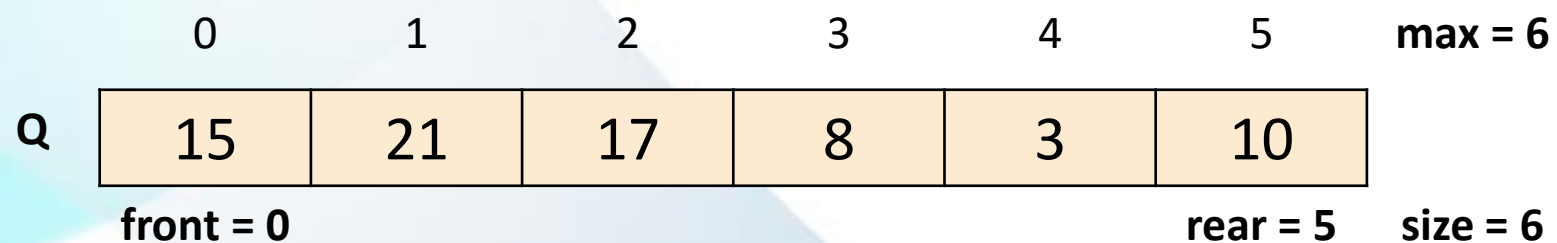
```
public void Create() {  
    Q = new int[max];  
    size = 0;  
    front = rear = -1;  
}
```

# Operasi IsFull

- Untuk mengecek apakah queue dalam kondisi **penuh** dengan cara memeriksa **size**
- Jika size sama dengan **max**, maka **full**
- Jika size masih lebih kecil dari **max**, maka belum **full**



# Operasi IsFull



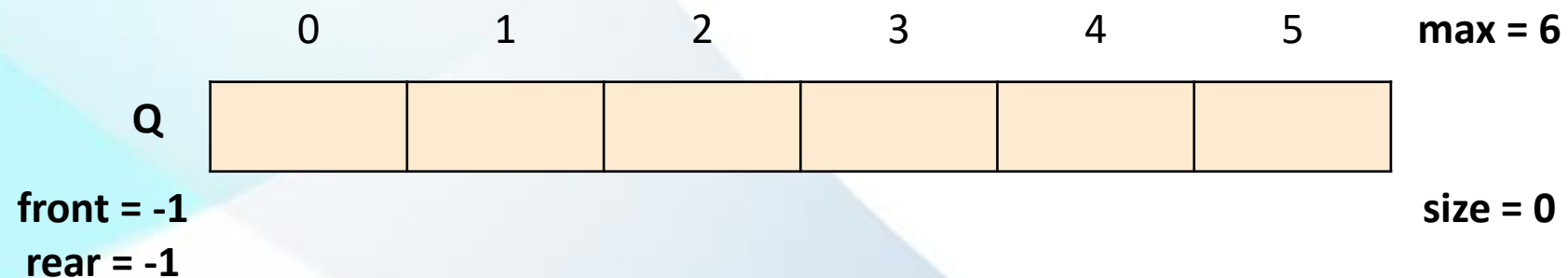
```
public boolean IsFull() {  
    if (size == max) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



# Operasi IsEmpty

- Untuk mengecek apakah queue dalam kondisi kosong dengan cara memeriksa **size**
- Jika size masih sama dengan 0, maka artinya stack masih **kosong**

# Operasi IsEmpty



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

# Operasi Peek

- Untuk mengakses elemen yang ditunjuk oleh front, yaitu elemen yang berada di posisi paling depan (tidak selalu berada pada indeks ke-0)

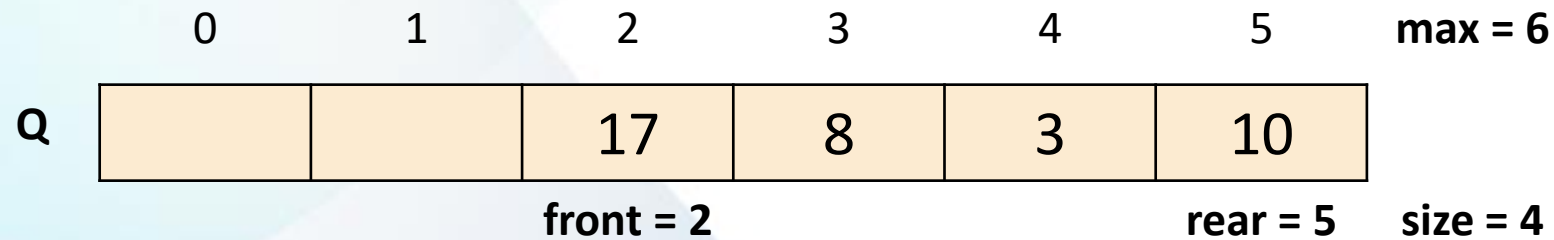
```
public void peek() {  
    if (!IsEmpty()) {  
        System.out.println("Elemen terdepan: " + Q[front]);  
    } else {  
        System.out.println("Queue masih kosong");  
    }  
}
```



# Operasi Print

- Untuk menampilkan semua data yang ada di dalam queue
- Proses dilakukan dengan cara me-loop semua isi array mulai dari indeks front sampai dengan indeks rear. Looping tidak selalu mulai dari indeks ke-0 karena front tidak selalu berada di indeks ke-0

# Operasi Print



Hasilnya: **17, 8, 3, 10**

```
public void print() {  
    if (IsEmpty()) {  
        System.out.println("Queue 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 elemen = " + size);  
    }  
}
```

# Operasi Clear

- Untuk menghapus elemen-elemen pada queue
- Penghapusan elemen-elemen tersebut dilakukan dengan mengeset indeks akses array (**front** dan **rear**) menjadi -1 agar elemen-elemen pada queue tidak dapat terbaca
- Variabel **size** juga perlu diset menjadi 0

```
public void clear() {  
    if (!IsEmpty()) {  
        front = rear = -1;  
        size = 0;  
        System.out.println("Queue berhasil dikosongkan");  
    } else {  
        System.out.println("Queue masih kosong");  
    }  
}
```

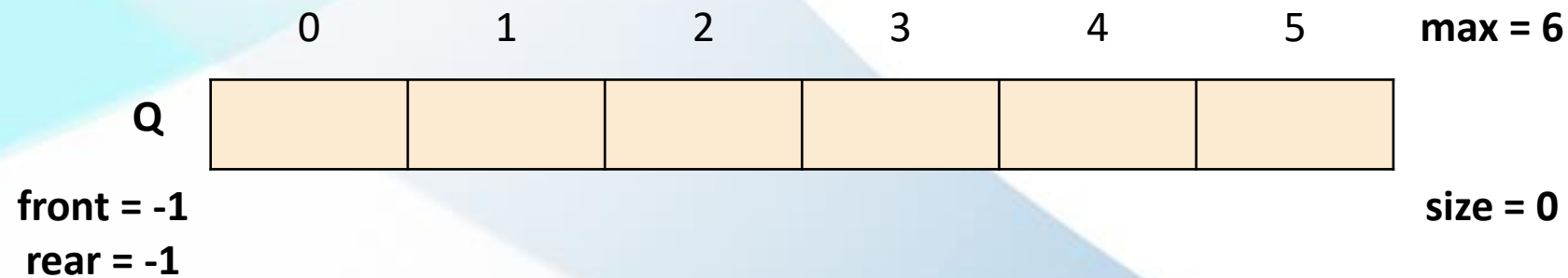
# Operasi Enqueue

- Untuk menambah data baru ke dalam queue
- Pada proses enqueue, data baru akan menempati **posisi paling akhir** dalam queue
- Terdapat 3 kemungkinan kondisi yang terjadi saat Enqueue:
  1. Ketika queue dalam kondisi kosong
  2. Ketika data paling belakang dari queue tidak berada di indeks terakhir array
  3. Ketika data paling belakang dari queue berada di indeks terakhir array



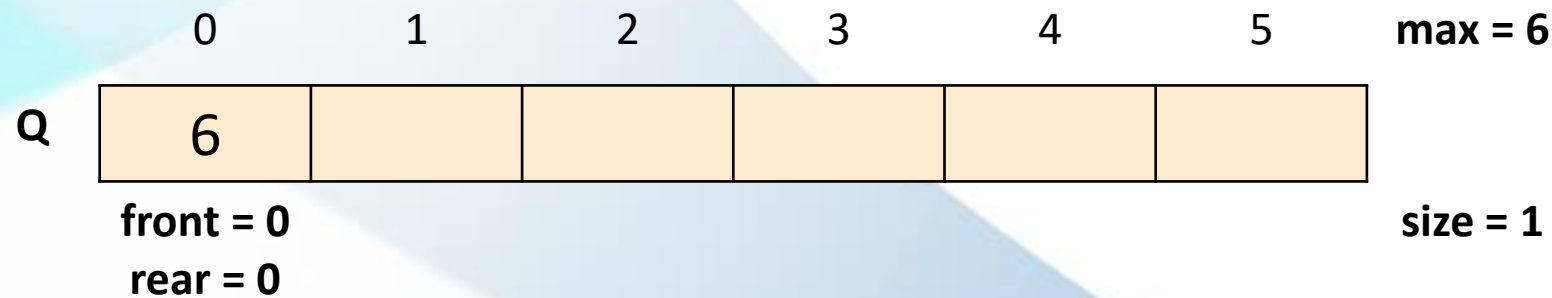
# Operasi Enqueue – Kondisi 1

## 1. Ketika queue dalam kondisi kosong



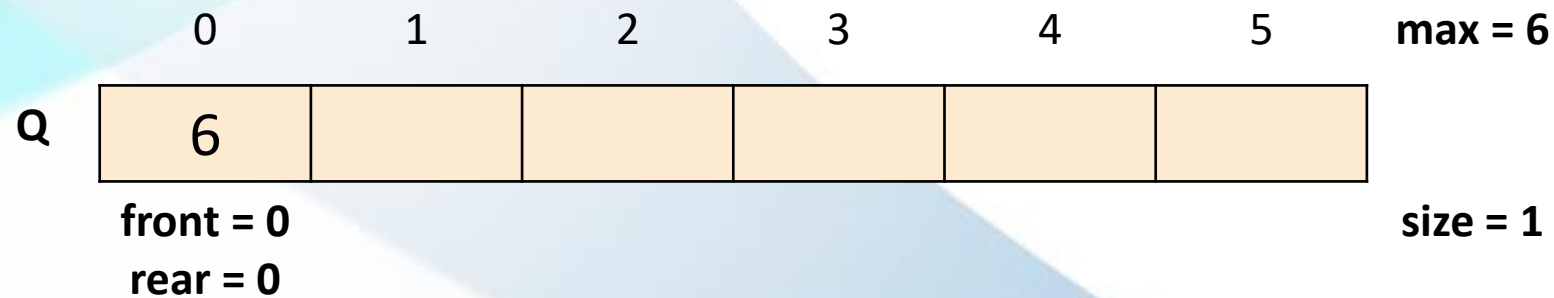
# Operasi Enqueue – Kondisi 1

- Ketika dilakukan penambahan data, maka data baru dimasukkan ke dalam queue pada indeks ke 0.
- Data tersebut menjadi data pada posisi FRONT dan REAR



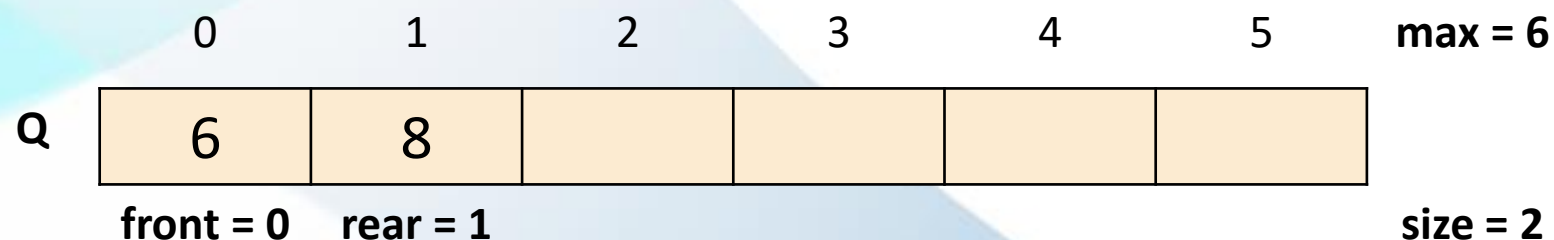
# Operasi Enqueue – Kondisi 2

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



## Operasi Enqueue – Kondisi 2

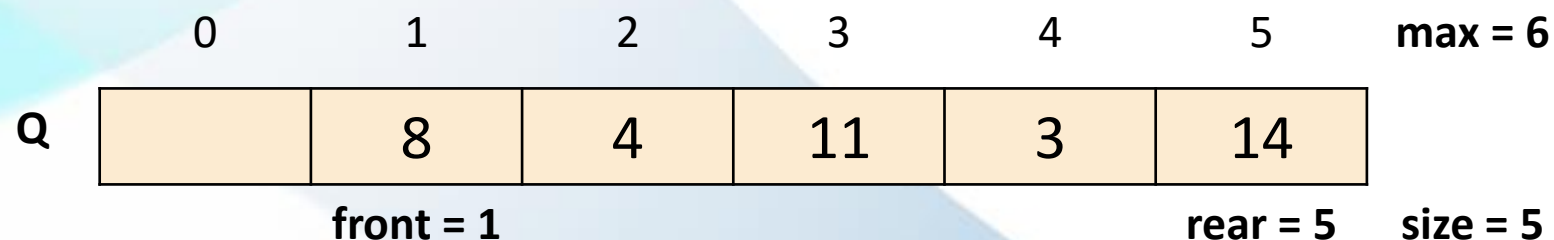
- Ketika dimasukkan data baru, maka data tersebut akan menempati posisi setelah data paling belakang saat ini, yaitu menempati indeks  $REAR + 1$



Awalnya rear = 0,  
ketika ada data baru masuk,  
maka rear =  $0 + 1 = 1$

## Operasi Enqueue – Kondisi 3

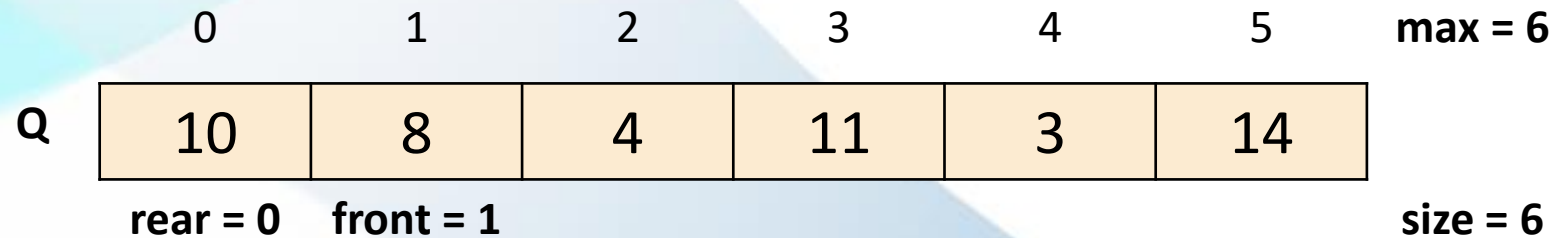
### 3. Ketika data paling belakang dari queue berada di indeks terakhir array



Perhatikan bahwa front tidak selalu berada pada indeks ke-0, bisa saja indeks ke-1 atau yang lain karena sebelumnya sudah ada data yang dikeluarkan

## Operasi Enqueue – Kondisi 3

- Ketika dimasukkan data baru, maka data tersebut akan menempati posisi indeks ke-0, artinya posisi REAR = 0



# Algoritma Operasi Enqueue

- Memastikan bahwa queue tidak dalam kondisi penuh. **Jika queue penuh**, maka data **tidak bisa** dimasukkan ke dalamnya.
- **Jika tidak penuh**, maka proses penambahan data bisa dilakukan.
  - Cek apakah queue dalam **kondisi kosong**. Jika 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  $FRONT = REAR = 0$
  - Jika queue dalam **kondisi tidak kosong**, kemudian:
    - Cek apakah posisi REAR berada pada indeks terakhir array. Jika benar, maka posisi REAR selanjutnya adalah di indeks 0
    - Jika posisi REAR tidak berada pada indeks terakhir array, maka posisi REAR selanjutnya adalah  $REAR + 1$
  - Masukkan data ke dalam queue pada indeks REAR
  - SIZE bertambah 1

# Algoritma Operasi Enqueue

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

Enqueue kondisi 1

Enqueue kondisi 3

Enqueue kondisi 2

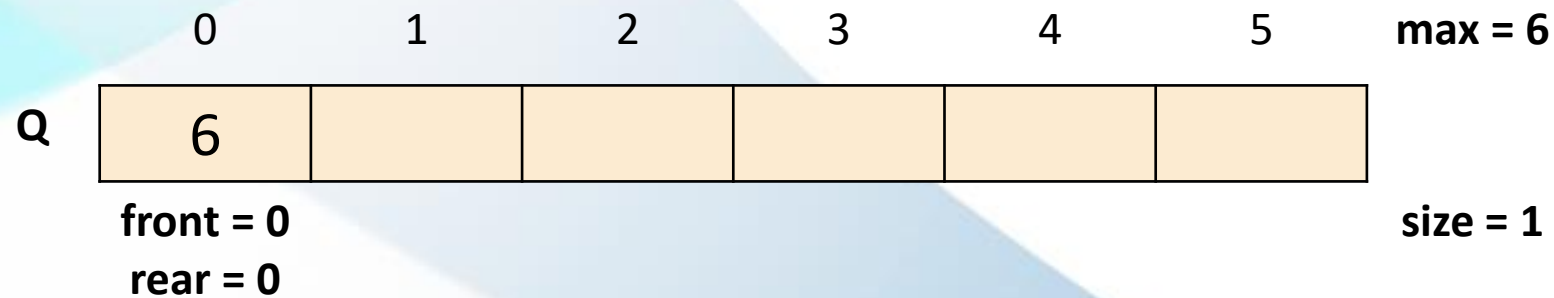


# Operasi Dequeue

- Untuk mengambil data dari queue
- Pada proses dequeue, data yang akan diambil adalah data yang menempati pada **posisi paling depan** (front) dalam queue
- Terdapat 3 kemungkinan kondisi yang terjadi saat Dequeue:
  - Ketika queue dalam kondisi kosong setelah data diambil
  - Ketika data yang paling depan dari queue tidak berada di indeks terakhir array
  - Ketika data paling depan dari queue berada di indeks terakhir array

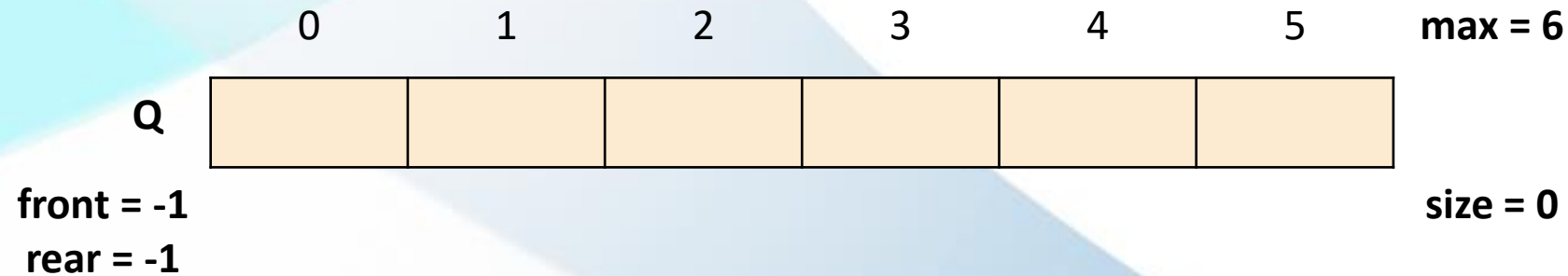
# Operasi Dequeue – Kondisi 1

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



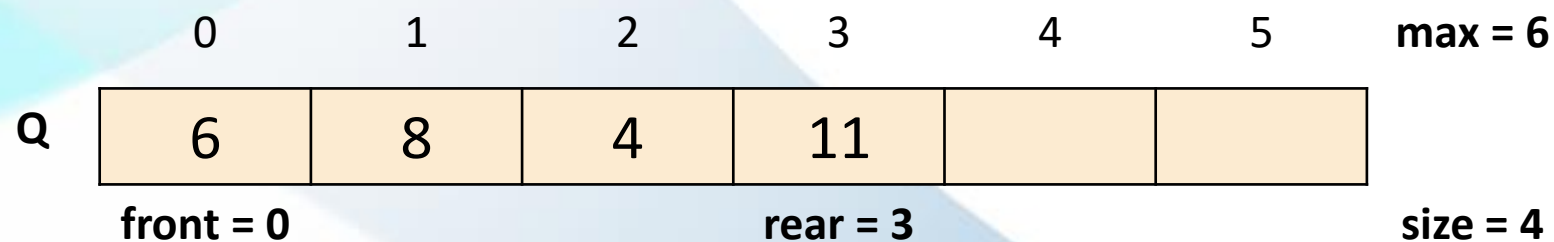
# Operasi Dequeue – Kondisi 1

- Ketika dilakukan pengambilan data, maka data yang diambil adalah 6, dan posisi FRONT dan REAR diset menjadi -1



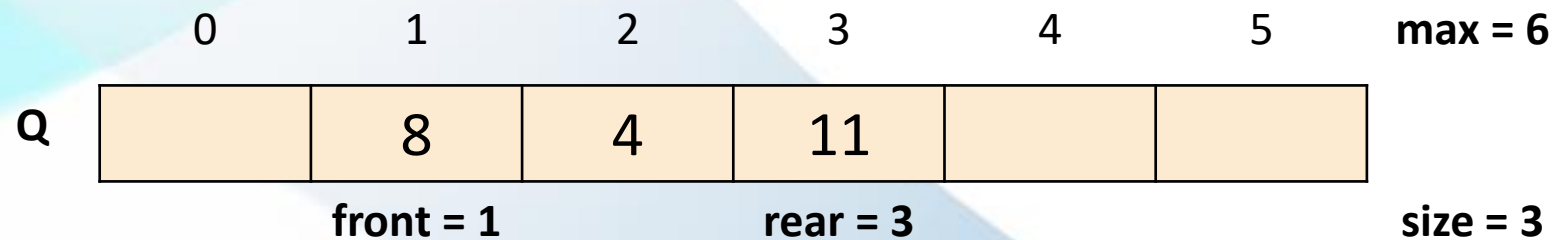
# Operasi Dequeue – Kondisi 2

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



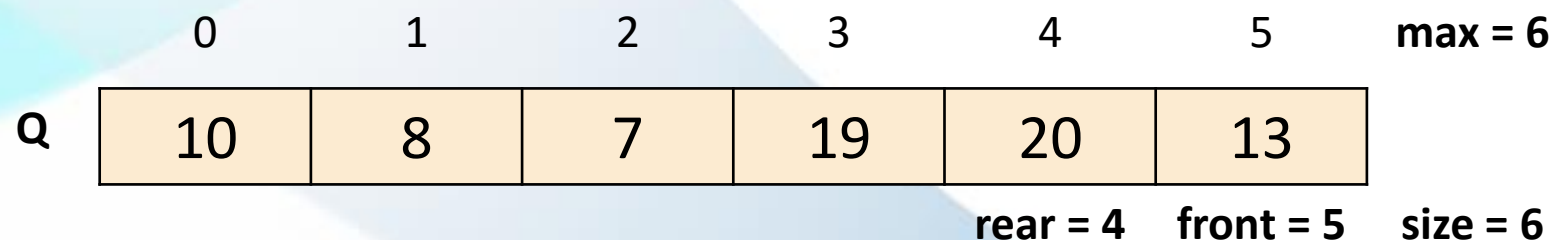
## Operasi Dequeue – Kondisi 2

- Ketika dilakukan pengambilan data, maka data yang diambil adalah 6, dan posisi FRONT akan bertambah 1 dari posisi sebelumnya



# Operasi Dequeue – Kondisi 3

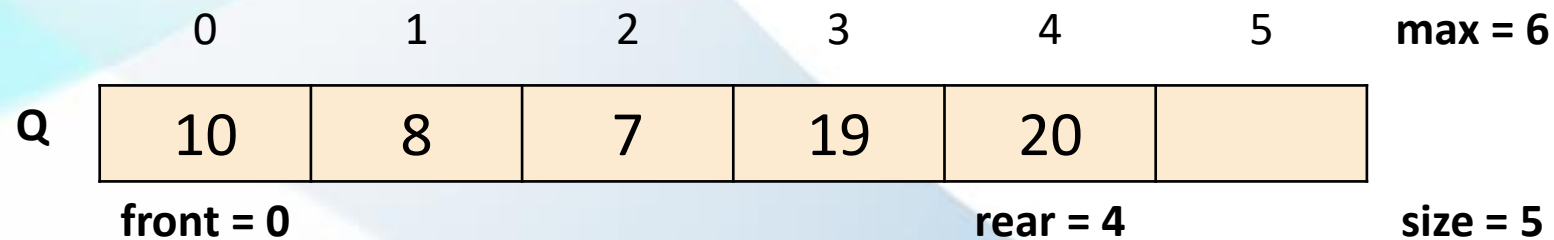
## 3. Ketika data paling depan dari queue berada di indeks terakhir array



Perhatikan bahwa indeks front bisa lebih besar dari rear karena pada kondisi penuh terdapat penghapusan data sampai front berada di indeks ke-5, kemudian dilakukan penambahan data sehingga menggeser indeks rear

## Operasi Dequeue – Kondisi 3

- Ketika dilakukan pengambilan data, maka data yang diambil adalah 15, dan posisi FRONT akan bergeser ke indeks ke-0



# Algoritma Operasi Dequeue

- Memastikan bahwa queue tidak dalam kondisi kosong. **Jika queue kosong**, maka tidak ada data yang bisa diambil
- **Jika tidak kosong**, maka proses pengambilan data dari queue bisa dilakukan.
  - Ambil data yang ada di indeks FRONT, dimana data tersebut akan di return-kan dari proses ini
  - SIZE berkurang 1
  - Selanjutnya, ubah posisi FRONT:
    - Cek apakah setelah diambil datanya, queue dalam **kondisi kosong** (SIZE = 0). Jika benar, maka posisi FRONT = REAR = -1
    - Jika setelah diambil datanya dan **queue tidak kosong**, kemudian:
      - Cek apakah posisi FRONT saat ini **berada di indeks terakhir array**. Jika benar, maka FRONT selanjutnya diletakkan di indeks 0
      - Jika posisi FRONT **tidak berada di indeks terakhir array**, maka posisi FRONT selanjutnya adalah FRONT sebelumnya ditambah 1



# Algoritma Operasi Dequeue

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

Dequeue kondisi 1

Dequeue kondisi 3

Dequeue kondisi 2

# Perubahan Front dan Rear

- Indeks rear bertambah 1 setiap kali terjadi Enqueue
- Indeks front bertambah 1 setiap kali terjadi Dequeue

- Kondisi sekarang: 

0	1	2	3	4	5
10	8	7			

front = 0 rear = 2 size = 3 max = 6
- Setelah Enqueue: 

0	1	2	3	4	5
10	8	7	19		

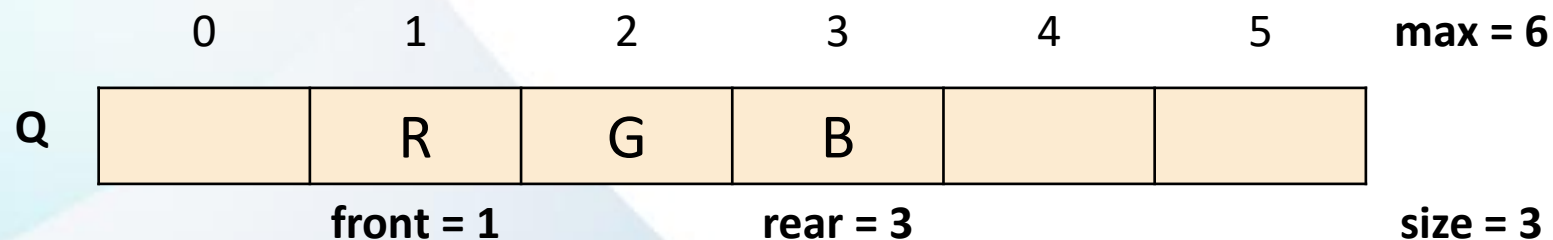
front = 0 rear = 3 size = 4 max = 6
- Setelah Dequeue: 

0	1	2	3	4	5
	8	7	19		

front = 1 rear = 3 size = 3

# Latihan

1. Terdapat Queue dengan kapasitas 6 elemen sebagai berikut:



Gambarkan kondisi Queue dan tentukan nilai rear dan front untuk beberapa operasi berikut:

- Menambahkan data A
- Menghapus data R dan G
- Menambahkan data X, Y, dan Z
- Menghapus data B dan A

2. Buatlah flowchart untuk operasi Enqueue dan Dequeue!