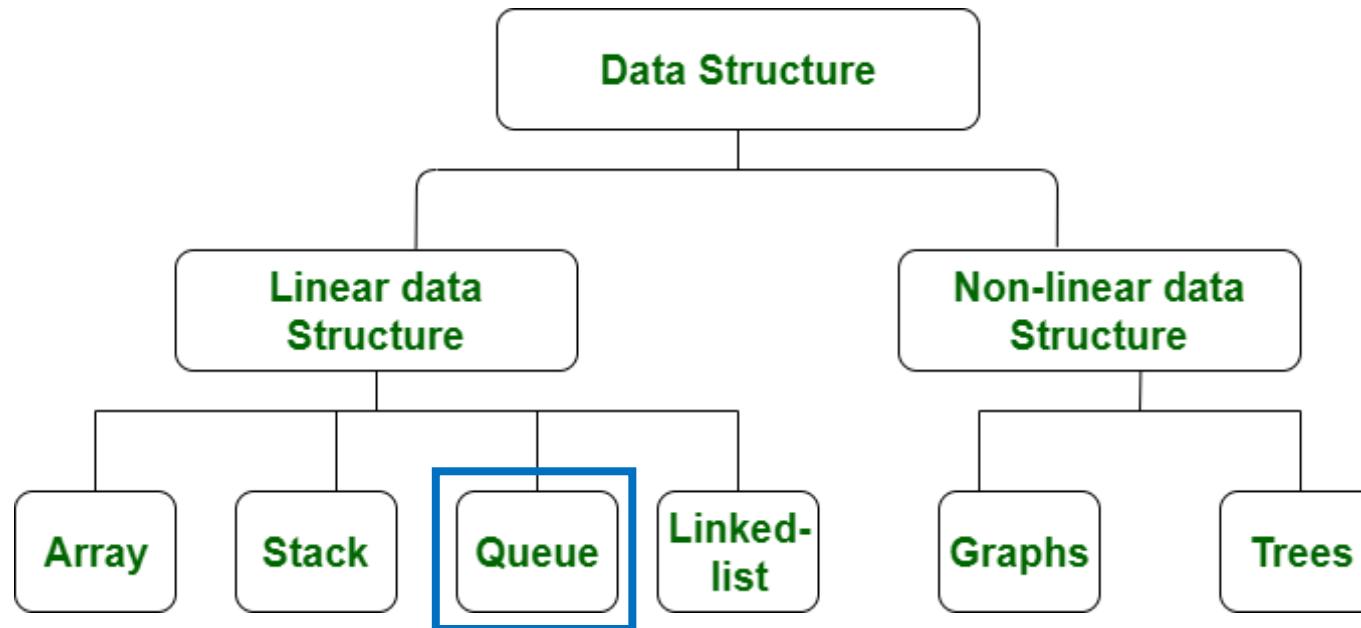




# QUEUE

Tim Ajar Algoritma dan Struktur Data  
Genap 2024/2025

# Jenis Struktur Data



# 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 Queue:
  - Barisan orang yang mengantre untuk membeli tiket, orang yang pertama datang akan dilayani terlebih dahulu
  - Antrian job di dalam sistem operasi

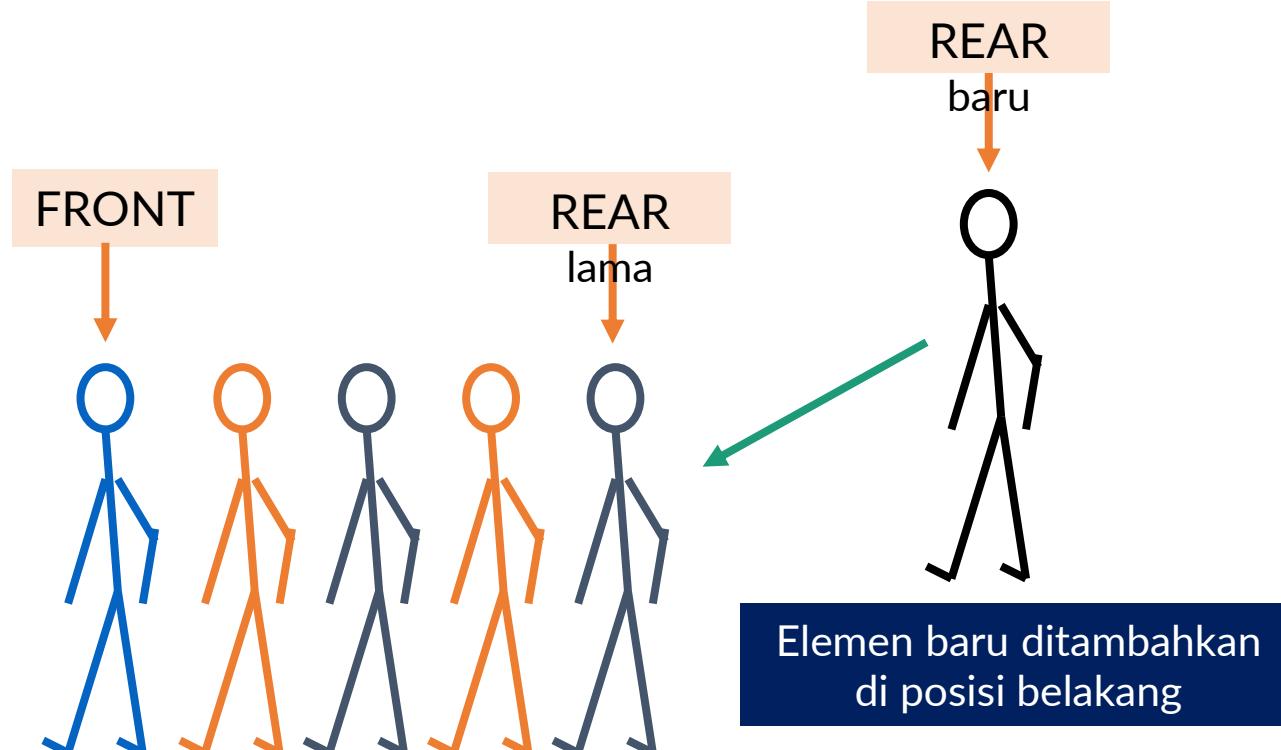
# Penerapan Queue

- Layanan permintaan pada single shared resource**  
Misalnya penggunaan printer, penjadwalan CPU, penjadwalan disk, dll
- Penanganan interrupt dalam real-time system**  
Interrupt ditangani sesuai dengan urutan (first come first served)
- Sistem Call Center**  
Menahan (hold) customer yang menelepon mereka secara berurutan
- Pada aplikasi perpesan (WhatsApp, Telegram, LINE, dll)**  
Urutan pesan diatur untuk setiap pengguna yang berisikan pesan yang akan dikirim. Saat pengguna terhubung ke jaringan, pesan di dalam queue akan terkirim

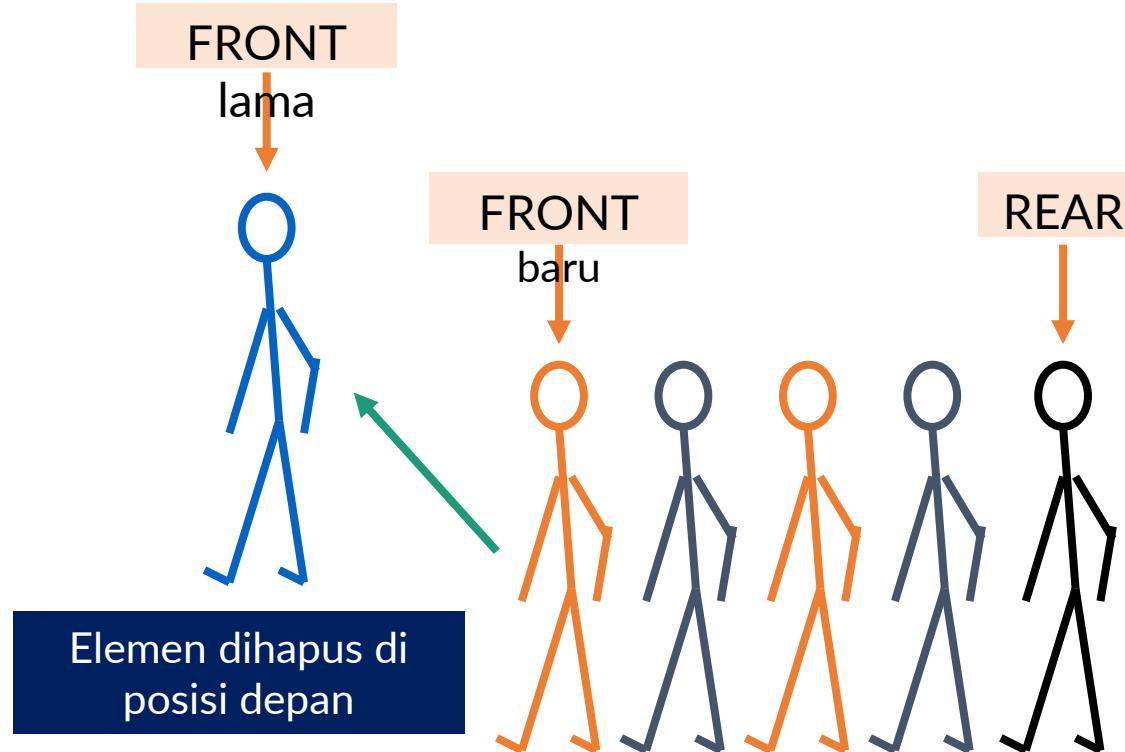
# Konsep Queue

- 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)

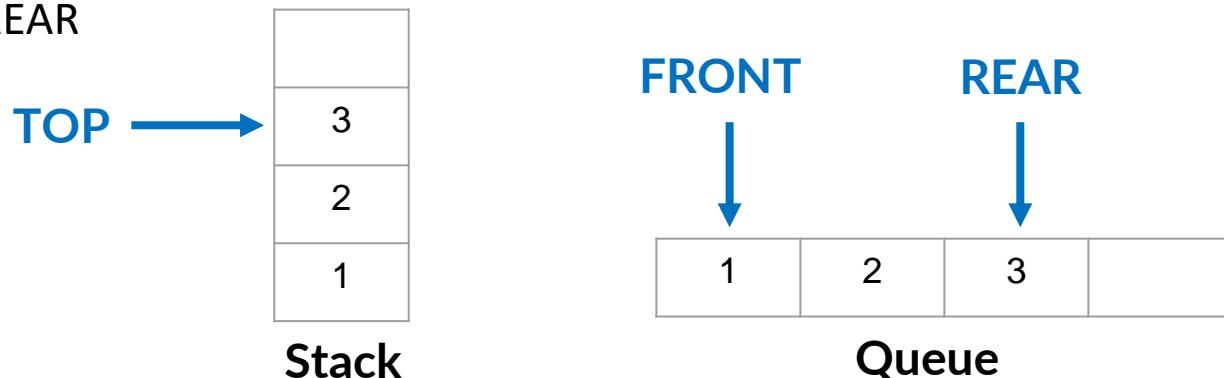


# Operasi Queue

1. **IsFull**: mengecek apakah queue dalam kondisi penuh
2. **IsEmpty**: mengecek apakah queue dalam kondisi kosong
3. **Enqueue**: menambah data dalam queue pada posisi paling belakang
4. **Dequeue**: mengambil data dari queue pada posisi paling depan
5. **Peek**: mengecek data paling depan
6. **Print**: menampilkan semua data pada queue
7. **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
- Penjelasan mengenai Linked List akan dibahas pada pertemuan berikutnya

# Cara Kerja Queue (Menggunakan Array)

Misalkan terdapat queue **data** dengan elemen sebanyak  $N$  ( $\text{data}_1, \text{data}_2, \dots, \text{data}_N$ )

1. Data di posisi **depan** queue disimbolkan **front(data)**
2. Data di posisi **belakang** queue  $Q$  disimbolkan **rear(data)**
3. **Jumlah elemen** di dalam queue dinyatakan dengan simbol **size(data)** yang dapat dihitung dengan dua cara berikut:
  - Jika  $\text{rear} \geq \text{front}$ : **rear – front + 1**
  - Jika  $\text{rear} < \text{front}$ : **max + rear - front + 1**
4. Untuk queue  $\text{data} = [\text{data}_1, \text{data}_2, \dots, \text{data}_N]$ , maka  
 $\text{front}(\text{data}) = \text{data}_1$   
 $\text{rear}(\text{data}) = \text{data}_N$   
 $\text{size}(\text{data}) = N$

# Cara Kerja Queue (Menggunakan Array)

1. **front**: variabel untuk menyimpan nilai indeks array data terdepan
2. **rear**: variabel untuk menyimpan nilai indeks array data paling belakang
3. **size**: variabel untuk menyimpan berapa banyak data yang ada dalam antrian
4. **max**: variabel untuk menyimpan banyak data maksimal yang bisa disimpan di dalam queue
5. **data**: variabel array untuk menyimpan data queue

# Cara Kerja Queue (Menggunakan Array)

- Ilustrasi ketika queue sudah penuh

	0	1	2	3	4	5	max = 6
data	6	5	9	18	12	7	
	front = 0					rear = 5	size = 6

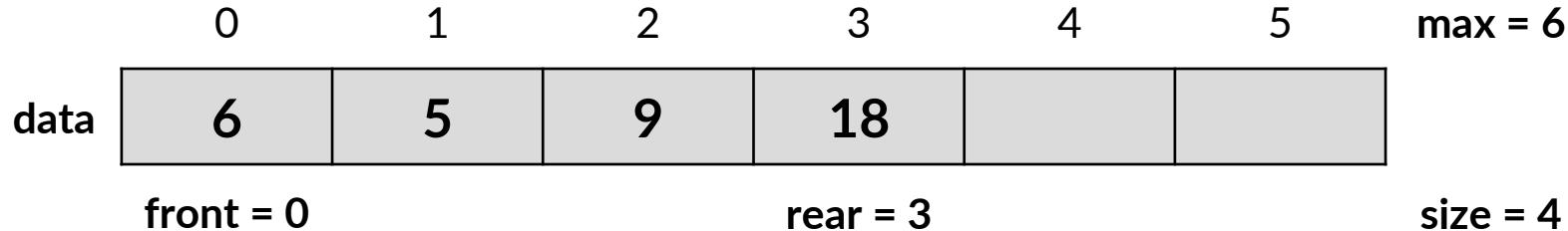
- Queue sudah terisi penuh dan tidak dapat menerima data lagi

## Queue overflow:

kondisi yang dihasilkan dari mencoba menambahkan elemen ke queue yang sudah penuh

# Cara Kerja Queue (Menggunakan Array)

- Ilustrasi ketika queue belum penuh



- ❑ Queue belum penuh sehingga masih dapat menerima data lagi

## Queue underflow:

kondisi yang dihasilkan dari mencoba menghapus elemen dari queue yang masih kosong

# Deklarasi Queue

- Proses pertama yang dilakukan adalah deklarasi atau menyiapkan tempat untuk queue
- Langkah-langkah:
  1. Deklarasi class
  2. Deklarasi atribut
    - a. Array data  
digunakan sebagai tempat penyimpanan data
    - b. front dan rear  
digunakan sebagai penunjuk data pada posisi depan dan belakang
    - c. size dan max  
digunakan untuk menentukan banyaknya data saat ini dan kapasitas penyimpanan

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

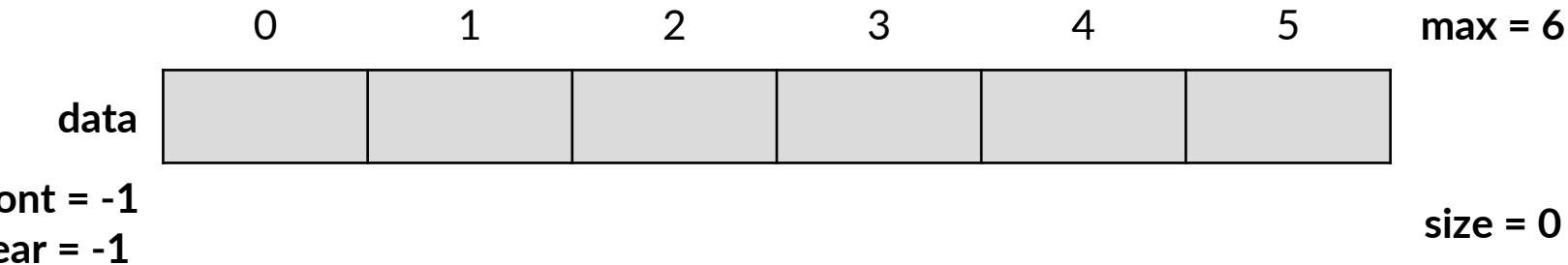
**Bounded Queue:**  
kapasitas queue ditentukan secara terbatas melalui konstruktor

# Inisialisasi 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 Queue – cont'd

- Ilustrasi queue saat inisialisasi pada konstruktor



```
public Queue(int n) {  
    max = n;  
    data = new int[max];  
    size = 0;  
    front = rear = -1;  
}
```

# Fungsi 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**

# Fungsi IsFull – cont'd

- #### □ Ilustrasi queue saat kondisi Full

A diagram illustrating a circular queue (array) with 6 slots. The slots are indexed from 0 to 5. The current state of the queue is as follows:

0	1	2	3	4	5	
data	15	21	17	8	3	10

Front pointer: **front = 0**

Rear pointer: **rear = 5**

Max size: **max = 6**

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

# Fungsi IsEmpty

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

# Fungsi IsEmpty – cont'd

- Ilustrasi queue saat kondisi kosong

0            1            2            3            4            5            **max = 6**



**front = -1**

**rear = -1**

**size = 0**

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

# Fungsi Peek

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

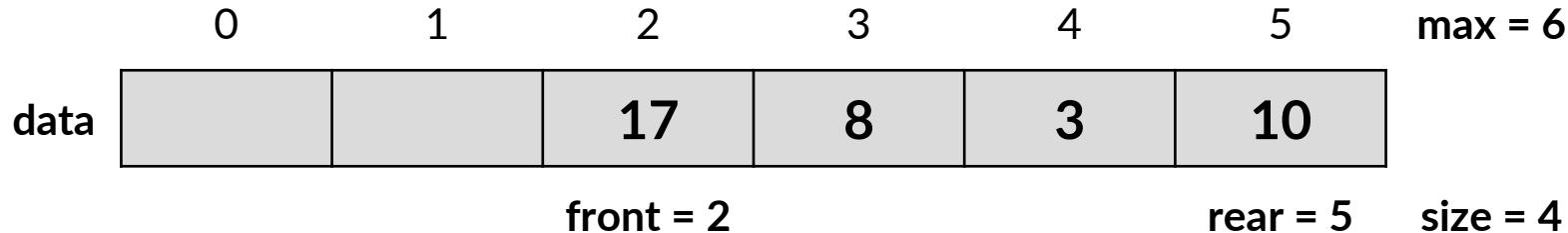
```
public void peek() {  
    if (!IsEmpty()) {  
        System.out.println("Elemen terdepan: " + data[front]);  
    } else {  
        System.out.println("Queue masih kosong");  
    }  
}
```

# Fungsi 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

# Fungsi Print



Hasil:

**17, 8, 3, 10**

Penyebab front tidak di posisi depan adalah Queue awalnya dalam keadaan penuh, kemudian dilakukan penghapusan elemen sehingga menyebabkan front bergeser ke belakang

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

# Fungsi 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");  
    }  
}
```



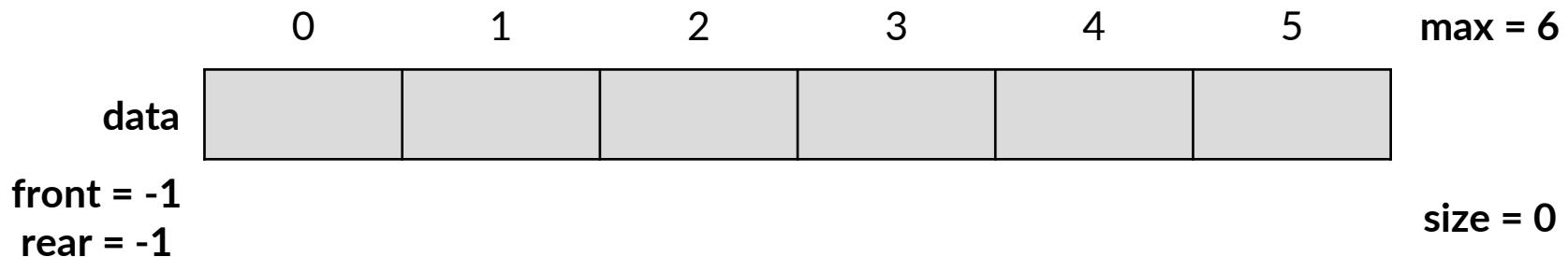
# Operasi Enqueue

# 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

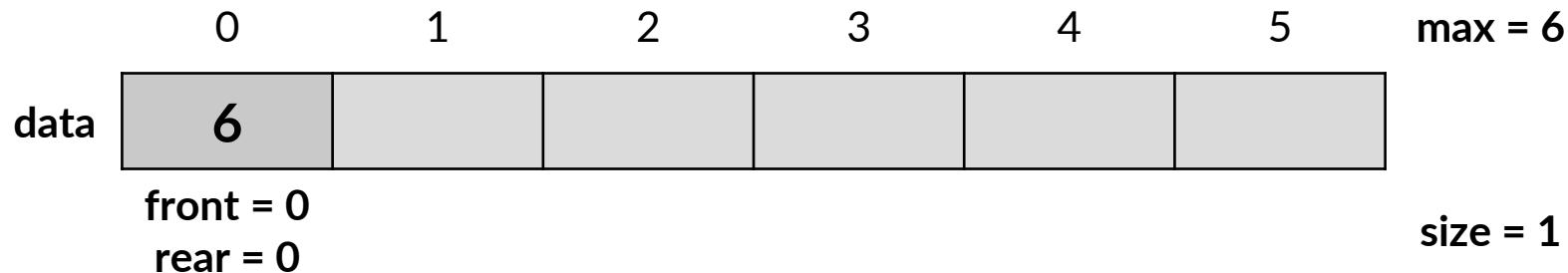
# Operasi Enqueue (Kondisi 1)

1. Ketika queue dalam kondisi kosong



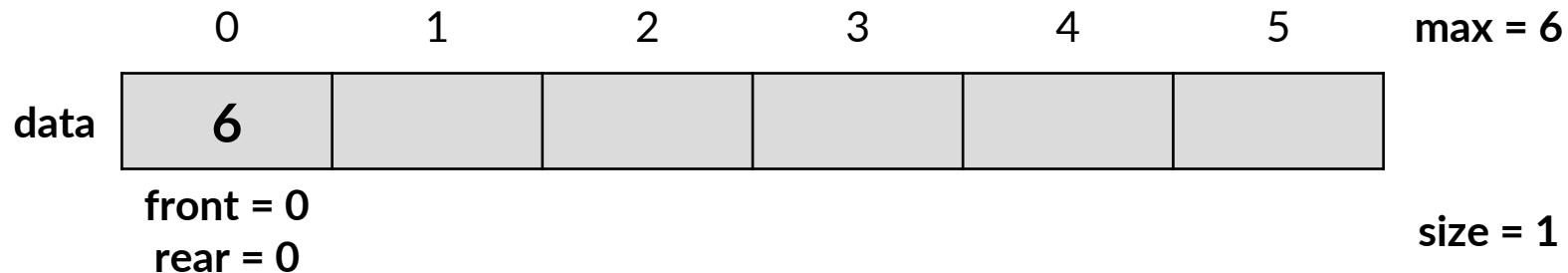
# Operasi Enqueue (Kondisi 1)

- ❑ 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**

	0	1	2	3	4	5	<b>max = 6</b>
<b>data</b>	6	8					
	<b>front = 0</b>	<b>rear = 1</b>					<b>size = 2</b>

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

A diagram illustrating a circular queue structure. At the top, indices 0 through 5 are labeled above a horizontal line. To the right of index 5, the text "max = 6" is displayed. Below this, the word "data" is followed by a horizontal box divided into six equal segments. The first segment is empty and shaded gray. The subsequent five segments contain the values 8, 4, 11, 3, and 14 respectively, also in gray. Below the box, the label "front = 1" is positioned under the second segment, "rear = 5" under the fifth segment, and "size = 5" to the right of the box.

	8	4	11	3	14
--	---	---	----	---	----

front = 1      rear = 5      size = 5

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

	0	1	2	3	4	5	<b>max = 6</b>
<b>data</b>	10	8	4	11	3	14	
	<b>rear = 0</b>	<b>front = 1</b>					<b>size = 6</b>

# Algoritma Enqueue

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 bisa dilakukan.
  - a. 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
  - b. Jika queue dalam **kondisi tidak kosong**, kemudian:
    - 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. Masukan data ke dalam queue pada indeks REAR
  - d. SIZE bertambah 1

# Algoritma Enqueue

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

Enqueue  
kondisi 1

Enqueue

kondisi 3

Enqueue  
kondisi 2

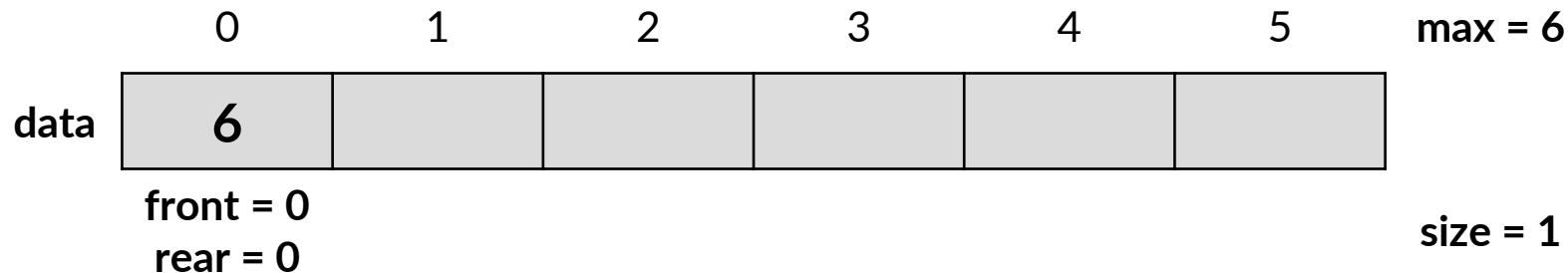
# Operasi Dequeue

# Operasi Dequeue

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

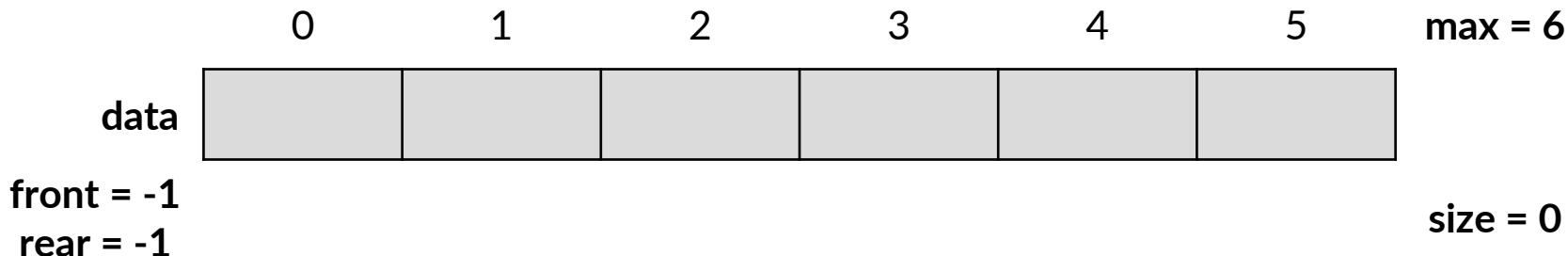
# Operasi Dequeue (Kondisi 1)

1. Ketika queue dalam kondisi kosong setelah data terambil



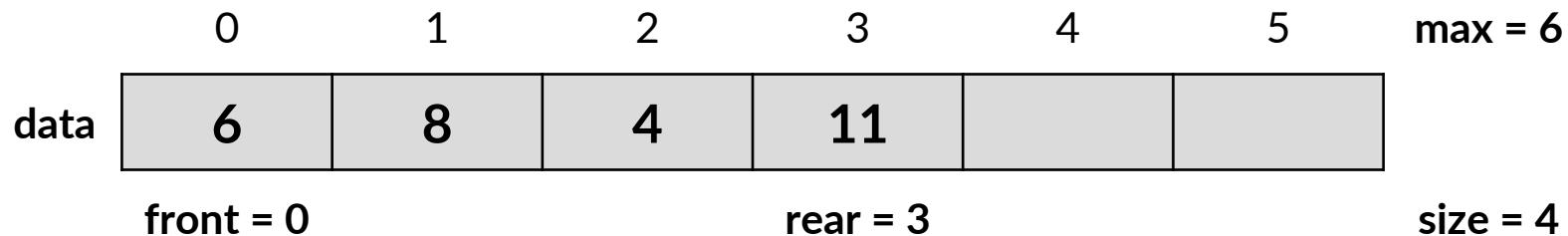
# Operasi Dequeue (Kondisi 1)

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



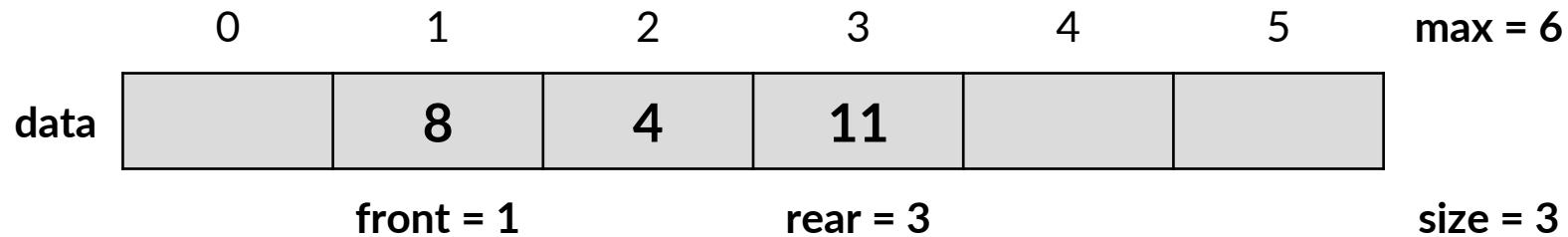
# Operasi Dequeue (Kondisi 2)

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



# Operasi Dequeue (Kondisi 2)

- Ketika dilakukan pengambilan data, maka data yang terambil 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

	0	1	2	3	4	5	max = 6
data	10	8	7	19	20	13	
	rear = 4		front = 5		size = 6		

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)

1. Ketika dilakukan pengambilan data, maka data yang terambil adalah 13, dan posisi **FRONT** akan bergeser ke indeks ke-0

A circular queue diagram with 6 slots. The first five slots contain the values 10, 8, 7, 19, and 20 respectively. The sixth slot is empty. Above the slots, indices 0 through 5 are shown, followed by **max = 6**. Below the slots, **data** is written to the left. At the bottom, **front = 0**, **rear = 4**, and **size = 5** are displayed.

0	1	2	3	4	5	max = 6
<b>data</b>	10	8	7	19	20	
	<b>front = 0</b>			<b>rear = 4</b>		<b>size = 5</b>

# Algoritma Dequeue

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** ( $\text{SIZE} = 0$ ). Jika benar, maka posisi  $\text{FRONT} = \text{REAR} = -1$
    - ii. 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 Dequeue

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

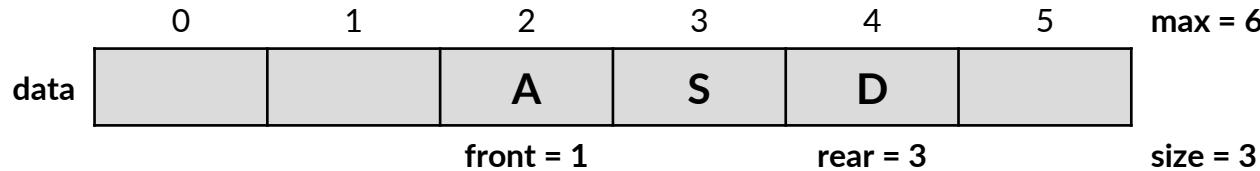
Dequeue  
kondisi 1

Dequeue  
kondisi 3

Dequeue  
kondisi 2

# Latihan

1. Terdapat Queue dengan kapasitas 6 elemen sebagai berikut:



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

- a. Menambahkan data P
  - b. Menghapus 2 data
  - c. Menambahkan data X, Y, dan Z
  - d. Menghapus 2 data
2. Buatlah flowchart untuk operasi Enqueue dan Dequeue!
  3. Tuliskan beberapa perbedaan antara Stack dan Queue dalam bentuk tabel!