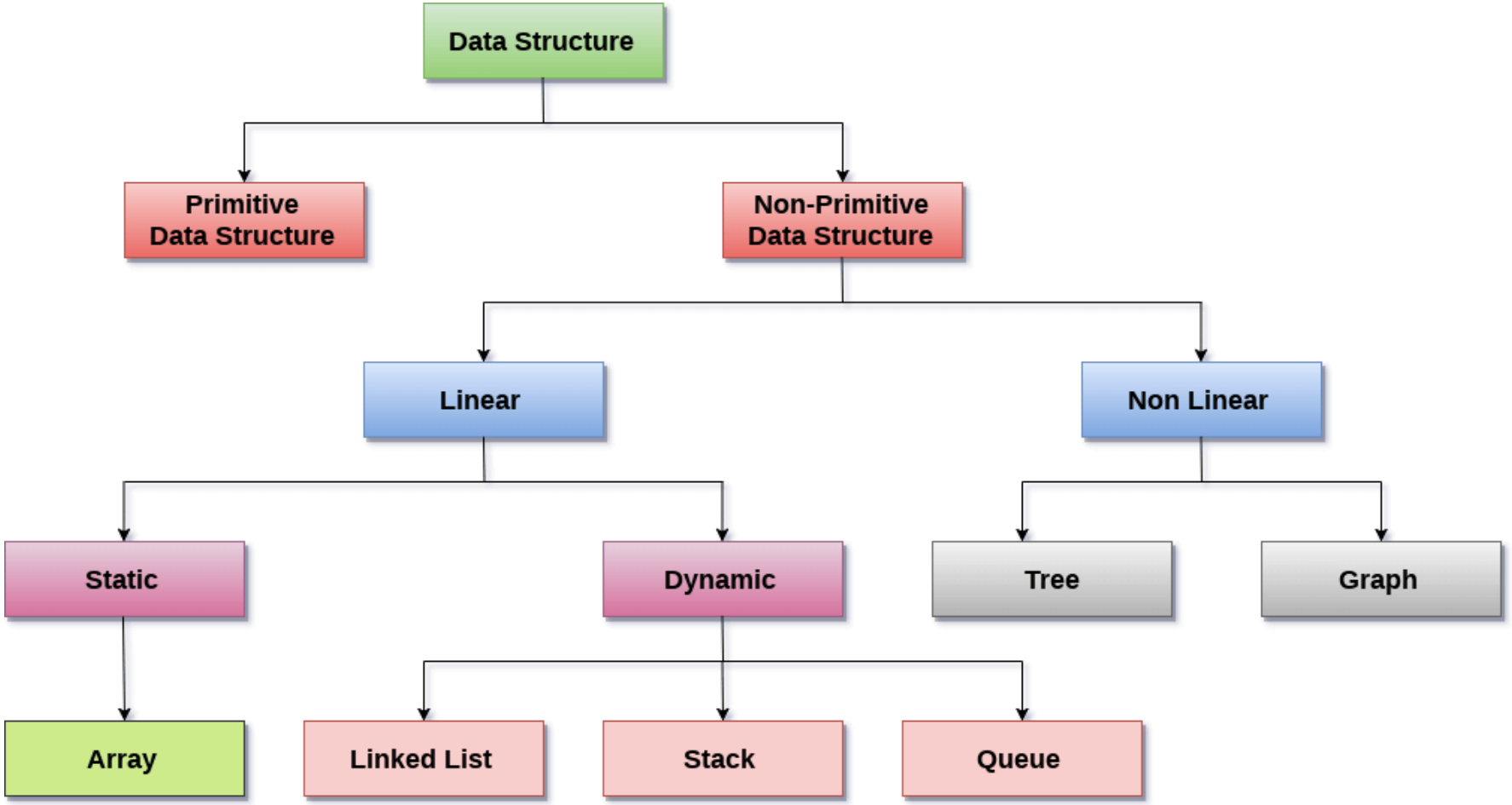
FAKULTAS TEKNOLOGI INFORMASI
Universitas Advent Indonesia

Lecture 08 – Queue/Antrian (Abstract Data Type)

Friday, November 3, 2023

Data Structure

Data Structures Classifications

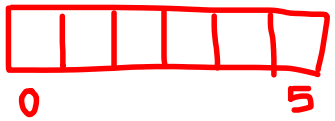


Introduction

- **Abstract Data Types (ADTs)** adalah cara mengklasifikasikan struktur data berdasarkan kegunaannya (bagaimana mereka digunakan) dan perilaku (behaviors) yang diberikan.

- Dalam hal ini, struktur data diklasifikasikan bukan dari bagaimana diimplementasikan namun hanya menyediakan antarmuka minimal yang diharapkan serta dengan perilakunya.

- Pembahasan mengenai struktur data sebagai ADT, berarti kita hanya akan membahas mengenai **fitur** atau **operasi-operasi** yang disediakan oleh struktur data tersebut. → **model logika**

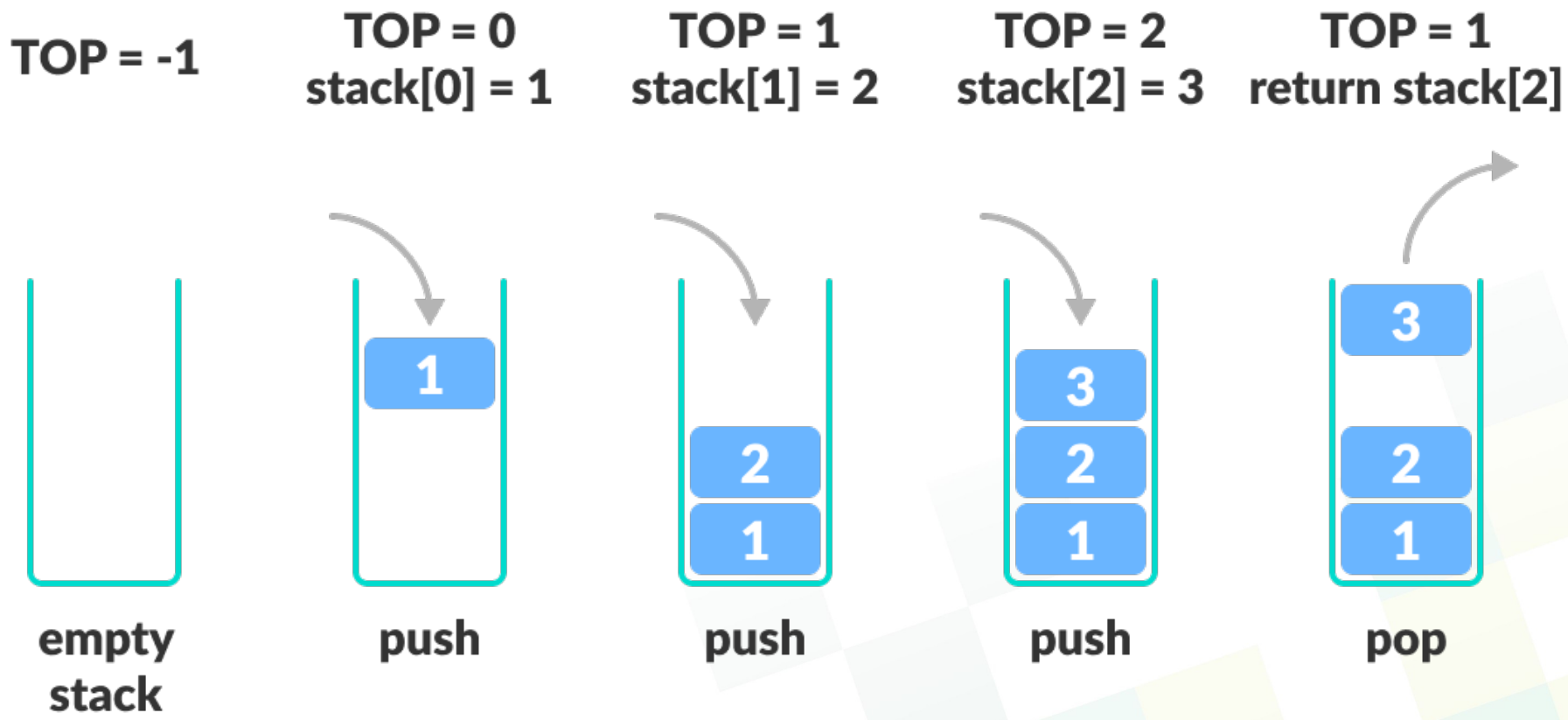


Stack as Abstract Data Types

- **Stack** dapat didefinisikan juga sebagai kumpulan elemen sama seperti Array dan List. Hal yang membedakan stack dari array atau list yakni elemen-elemen di dalam stack hanya dapat dimanipulasi dari satu bagian/ujung. Bagian/ujung ini umumnya dikenal dengan `top of stack`.

- Stack bekerja dengan prinsip **LIFO** (**Last In First Out**). Elemen terakhir yang dimasukkan ke dalam stack akan menjadi elemen pertama yang dikeluarkan.

How Stack works



Queue as Abstract Data Types

- **Queue** dapat dikatakan sebagai sebuah array atau list dimana elemen-elemen yang berada dalam queue ditambahkan/masuk melalui satu ujung yang disebut dengan **rear** (**tail**) dan dihapus/keluar melalui ujung lainnya yang disebut dengan **front** (**head**).

- Berbeda dengan stack yang bekerja dengan prinsip LIFO (Last In First Out), Queue bekerja dengan prinsi **FIFO** (**First In First Out**). Elemen pertama yang masuk ke dalam queue akan menjadi elemen pertama yang dikeluarkan.

Queue as Abstract Data Types

Queue as Abstract Data Types

Queue Operations

Operasi-operasi yang berlaku dalam Queue adalah :

- **Enqueue** : menambahkan elemen ke bagian belakang queue
- **Dequeue** : mengeluarkan elemen dari bagian depan queue
- **IsEmpty** : mencari tahu apakah queue kosong
- **IsFull** : mencari tahu apakah queue sudah penuh
- **Peek** : mengambil nilai dari elemen paling depan (front) dari queue tanpa mengeluarkan atau menghapus elemen tersebut.

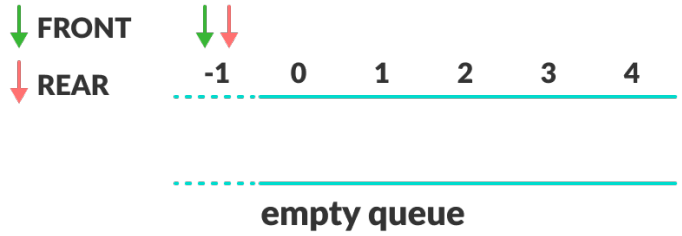
How Queue works

1. Dua variabel yang dikenal dengan **Front** (**head**) dan **Rear** (**tail**) digunakan sebagai variable untuk melacak elemen pertama dan terakhir dalam queue.
2. Inisialisasi queue dengan mengatur **front** dan **rear = -1**.
3. Sebelum melakukan enqueue, cek terlebih dahulu apakah queue sudah penuh.
4. Pada saat sebuah elemen baru masuk ke dalam queue (enqueue), nilai index dari rear akan bertambah dan elemen baru tersebut akan ditempatkan di posisi yg ditunjuk oleh rear.
5. Ketika melakukan enqueue elemen pertama, nilai dari front diatur menjadi 0

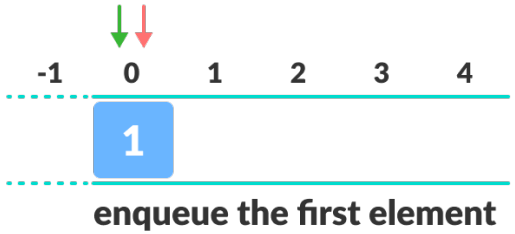
How Queue works

6. Sebelum melakukan dequeue, cek terlebih dahulu apakan queue kosong.
7. Pada saat mengeluarkan suatu elemen dari queue (dequeue), nilai front akan diambil dan indeksnya akan bertambah.
8. Ketika melakukan dequeue elemen terakhir, reset nilai dari front dan rear menjadi -1.

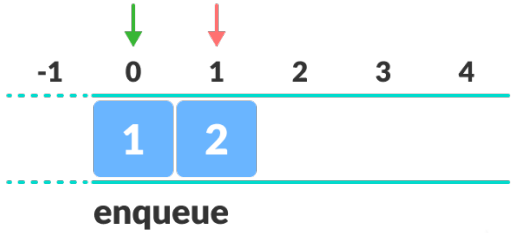
How Queue works



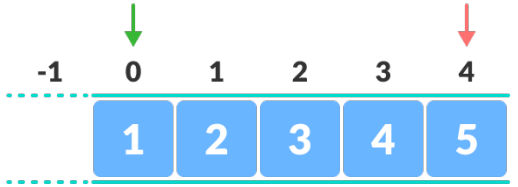
empty queue



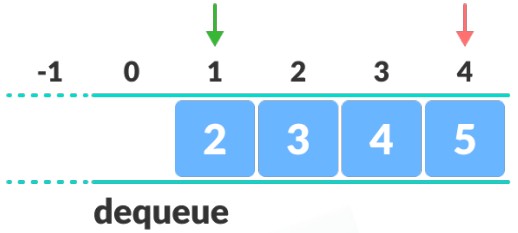
enqueue the first element



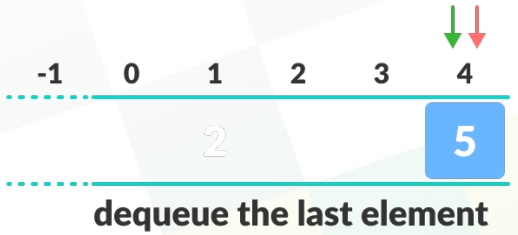
enqueue



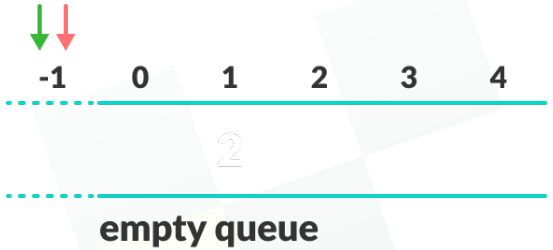
enqueue



dequeue



dequeue the last element



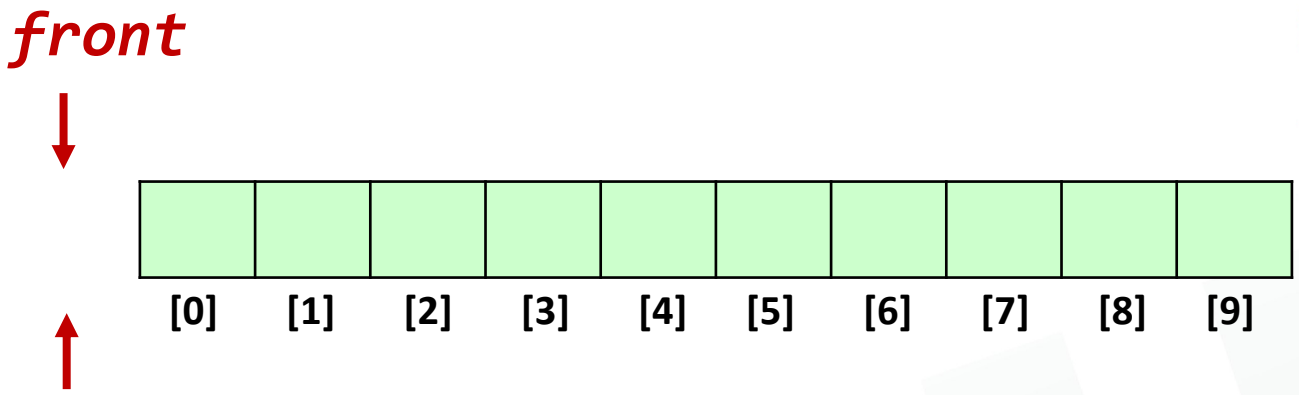
empty queue

Queue Applications

- Printer queue
- Penjadwalan CPU, Penjadwalan Disk
- Queue digunakan untuk sinkronisasi ketika transfer data antara dua proses terjadi secara asynchronous.
  Misalnya: IO Buffer, pipa, file IO, dll
- Penanganan interupsi dalam real-world system.
- Sistem telepon Call Center menggunakan Antrian untuk membuat orang memanggil mereka secara berurutan

Queue Implementation using Array

```
int  a[10];
front ← -1
rear ← -1
```

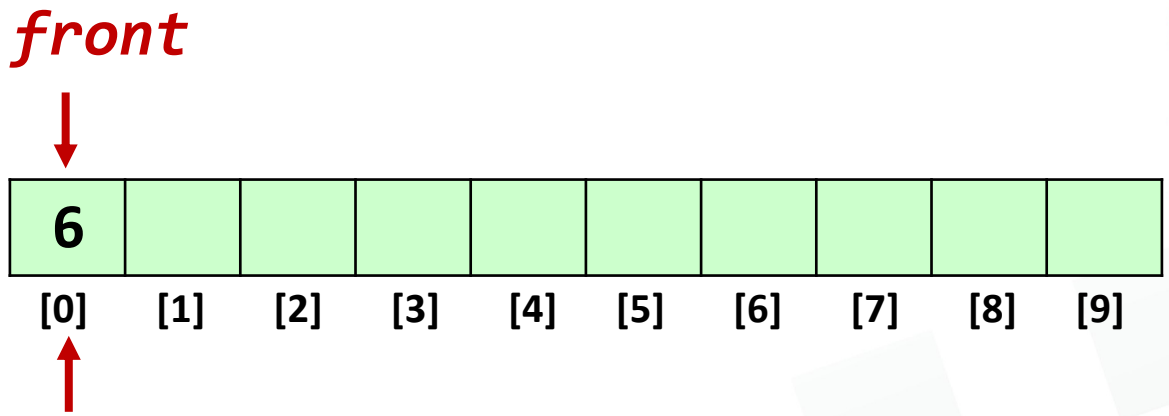



```
isEmpty()
{
    if front == -1 && rear ==
-1
        return true
    else
        return false
}
```

```
isFull()
{
    if rear == size(a)-1
        return true
    else
        return false
}
```

Queue Implementation using Array

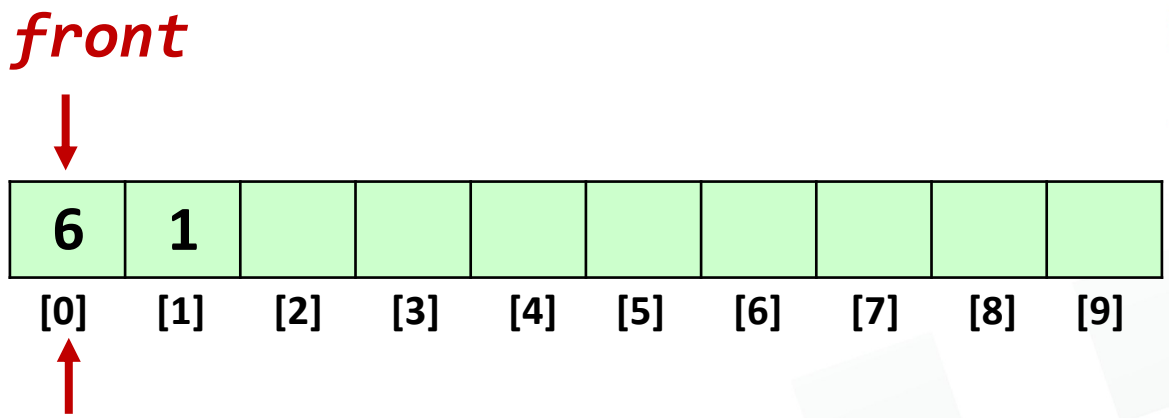
```
int  a[10];
front ← -1
rear ← -1
Enqueue(x)
{
   if isFull()
          return
   else if isEmpty() {
          front ← rear ← 0
   }
   else {
          rear ← rear + 1
   }
   a[rear] ← x
}
```



Enqueue (6)

Queue Implementation using Array

```
int  a[10];
front ← -1
rear ← -1
Enqueue(x)
{
   if isFull()
          return
   else if isEmpty() {
          front ← rear ← 0
   }
   else {
          rear ← rear + 1
   }
   a[rear] ← x
}
```

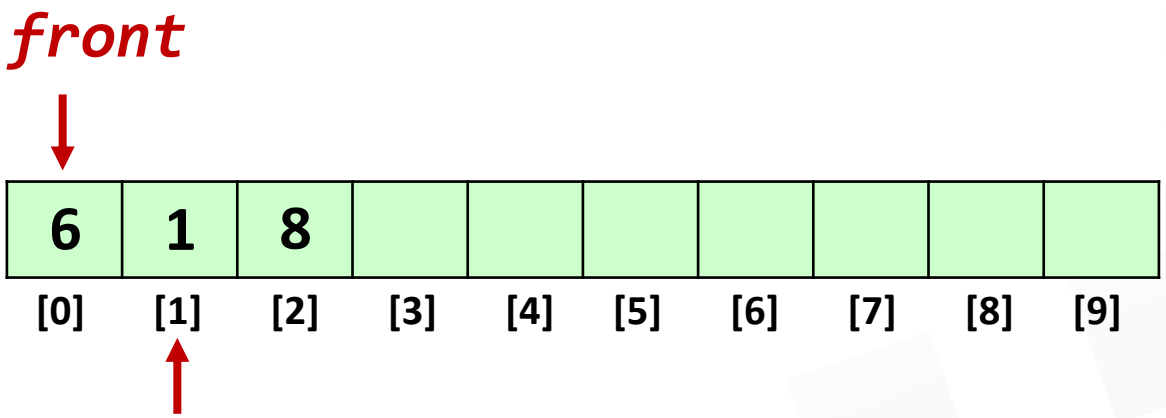


Enqueue (6)

Enqueue (1)

Queue Implementation using Array

```
int  a[10];
front ← -1
rear ← -1
Enqueue(x)
{
   if isFull()
          return
   else if isEmpty() {
          front ← rear ← 0
   }
   else {
          rear ← rear + 1
   }
   a[rear] ← x
}
```



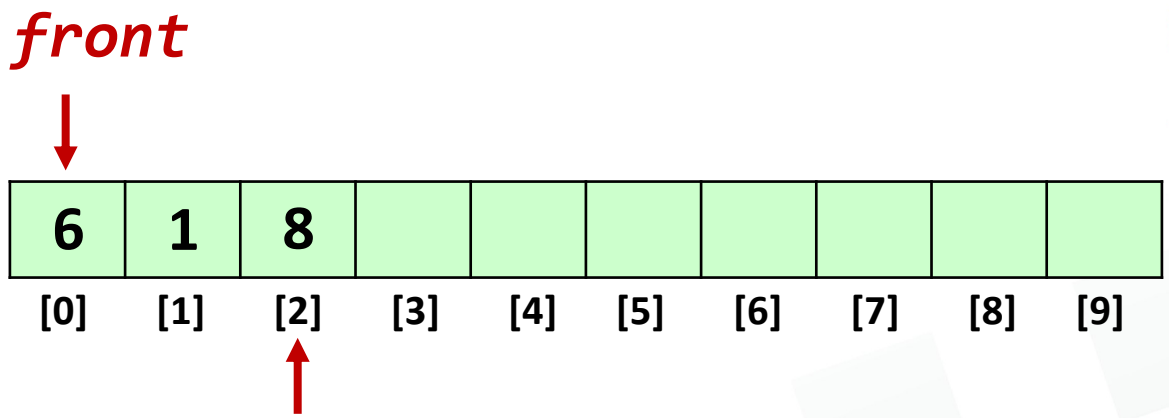
Enqueue (6)

Enqueue (1)

Enqueue (8)

Queue Implementation using Array

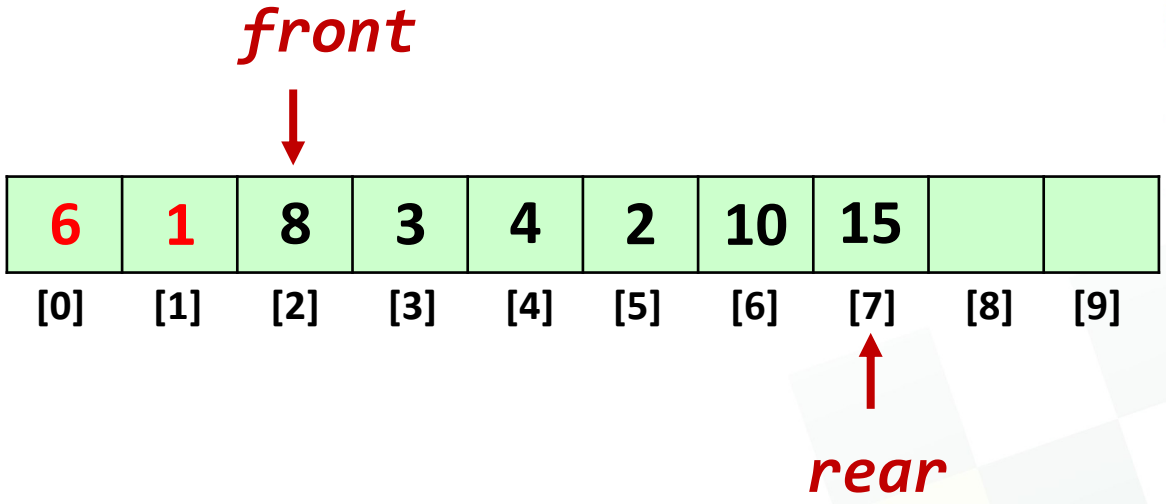
```
Dequeue()
{
    if isEmpty()
        return
    else if front==rear
        front ← rear ← -1
    else
        front ← front + 1
}
```




```
Enqueue(6)
Enqueue(1)
Enqueue(8)
Dequeue()
```

Queue Implementation using Array

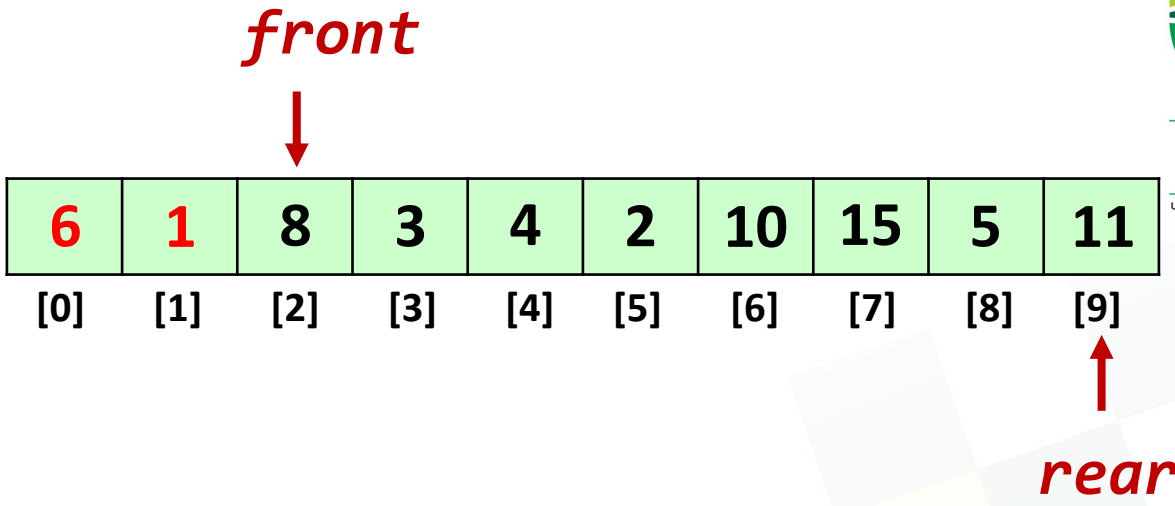
```
Dequeue()
{
    if isEmpty()
        return
    else if front==rear
        front ← rear ← -1
    else
        front ← front + 1
}
```




```
Enqueue(6)      Enqueue(4)
Enqueue(1)      Enqueue(2)
Enqueue(8)      Enqueue(10)
Dequeue()       Enqueue(15)
Enqueue(3)      Dequeue()
```

Queue Implementation using Array

```
Dequeue()
{
    if isEmpty()
        return
    else if front==rear
        front ← rear ← -1
    else
        front ← front + 1
}
```



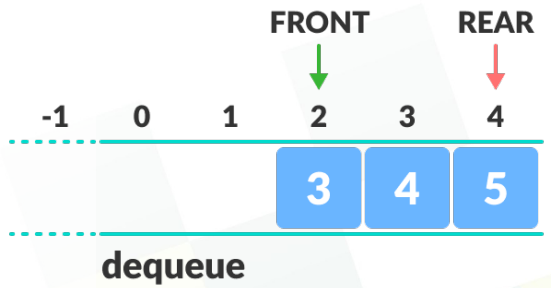

```
Enqueue(6)
Enqueue(1)
Enqueue(8)
Dequeue()
Enqueue(3)
```

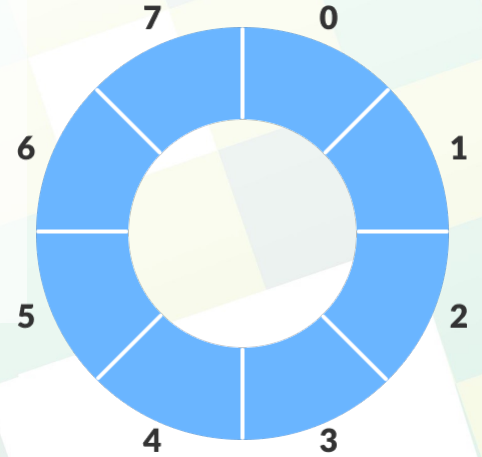
```
Enqueue(4)
Enqueue(2)
Enqueue(10)
Enqueue(15)
Dequeue()
```

```
Enqueue(5)
Enqueue(11)
```

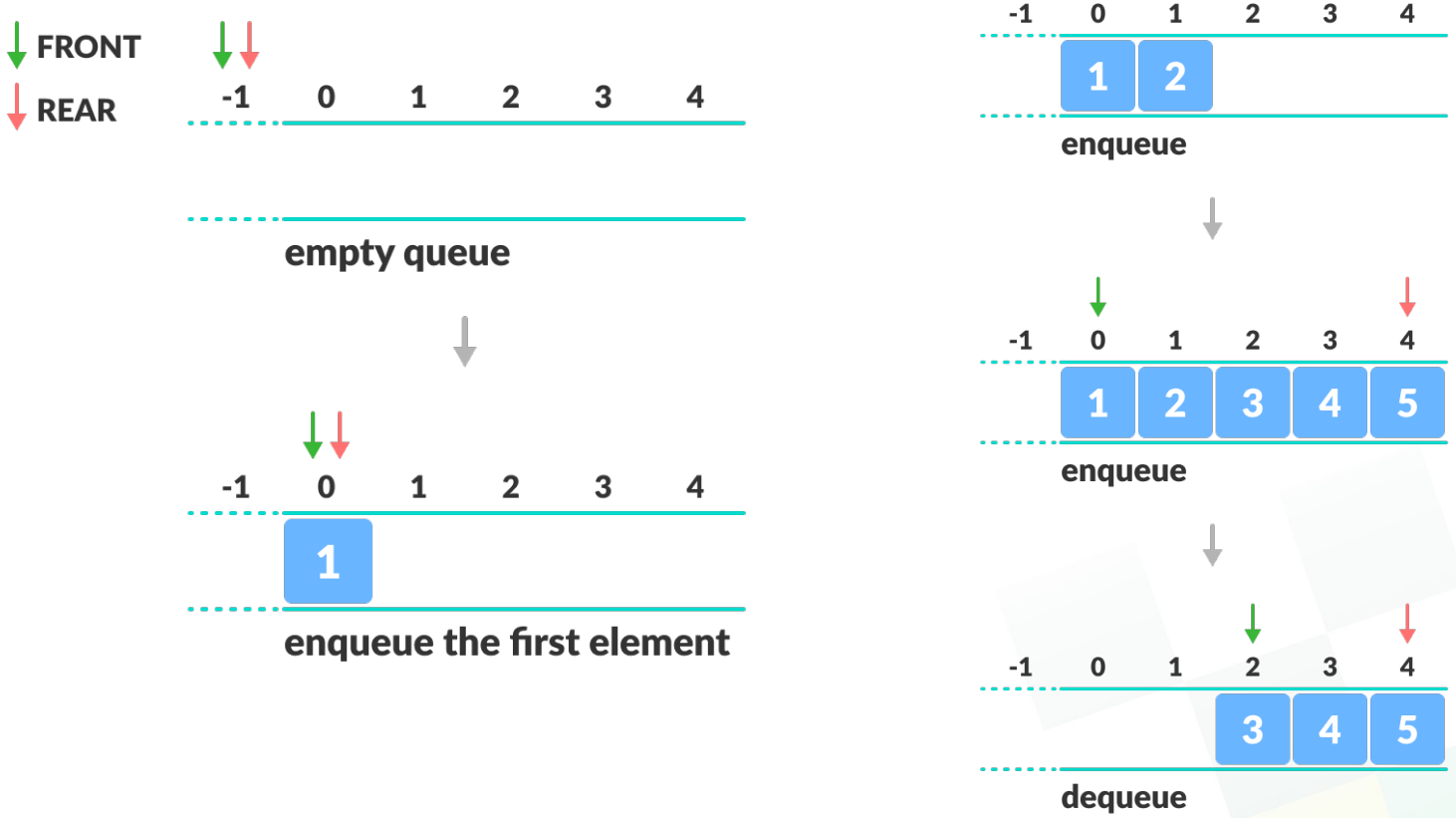
Circular Queue

- Circular queue digunakan untuk menghindari pemborosan ruang yang terjadi dalam implementasi queue menggunakan array.
- Dalam implementasi queue menggunakan array:
  - Current position = i
  - Next position = i + 1
- Dalam circular queue:
  - Current position = i
  - Next position = (i + 1) % N (*N adalah ukuran array)
  - Previous position = (i + N – 1) % N





How Circular Queue Works



Queue using STL

- Dalam C++, terdapat STL yang menyediakan fungsionalitas struktur data queue.

STL Queue Methods

- Di C++, class queue menyediakan berbagai method untuk melakukan operasi yang berbeda pada queue

Methods	Description
push()	inserts an element at the back of the queue
pop()	removes an element from the front of the queue
front()	returns the first element of the queue
back()	returns the last element of the queue
size()	returns the number of elements in the queue
empty()	returns true if the queue is empty