「不二坊」再現排隊人龍 警加強拖吊

What will you do to process the long line?

https://news.ltn.com.tw/news/life/paper/1473992 2021/09/21 05:30 自由時報

Queues

Ch 3.3 & Ch 3.4

Queues

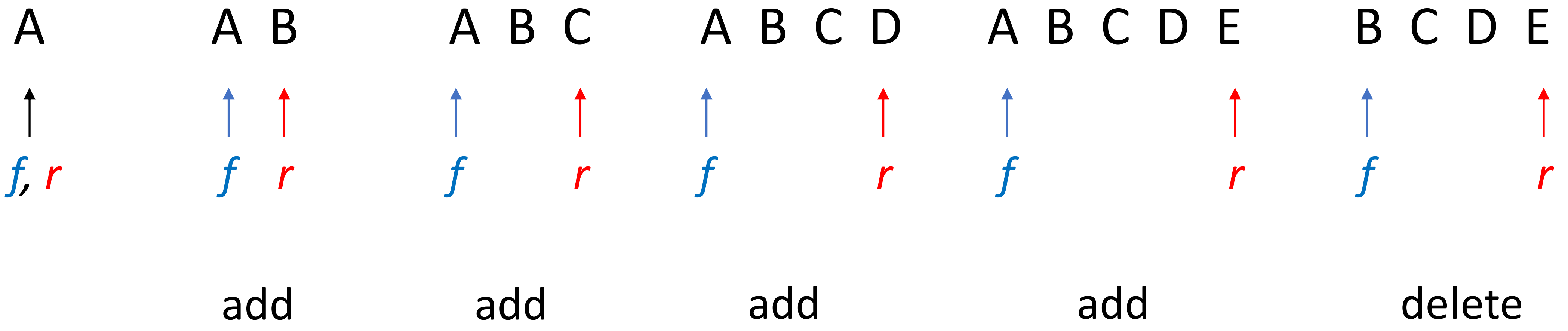
- An ordered list.
- Insertions and deletions take place at different ends.
- New elements are added at the rear end.
- Old elements are deleted at the front end.

- The first element inserted into a queue is the first element removed.

First-In-First-Out (FIFO)

Insertion and deletion of elements

- *f*: front of the queue
- *r*: rear of the queue



A	A B	A B C	A B C D	A B C D E	B C D E
f, *r*	*f* *r*	*f* *r*	*f* *r*	*f* *r*	*f* *r*
	add	add	add	add	delete

Operations of queues

- **CreateQ**: create an empty queue and set up variables *rear* and *front* as -1.
- **IsFullQ**: return True if queue is full

  rear == MAX_QUEUE_SIZE - 1

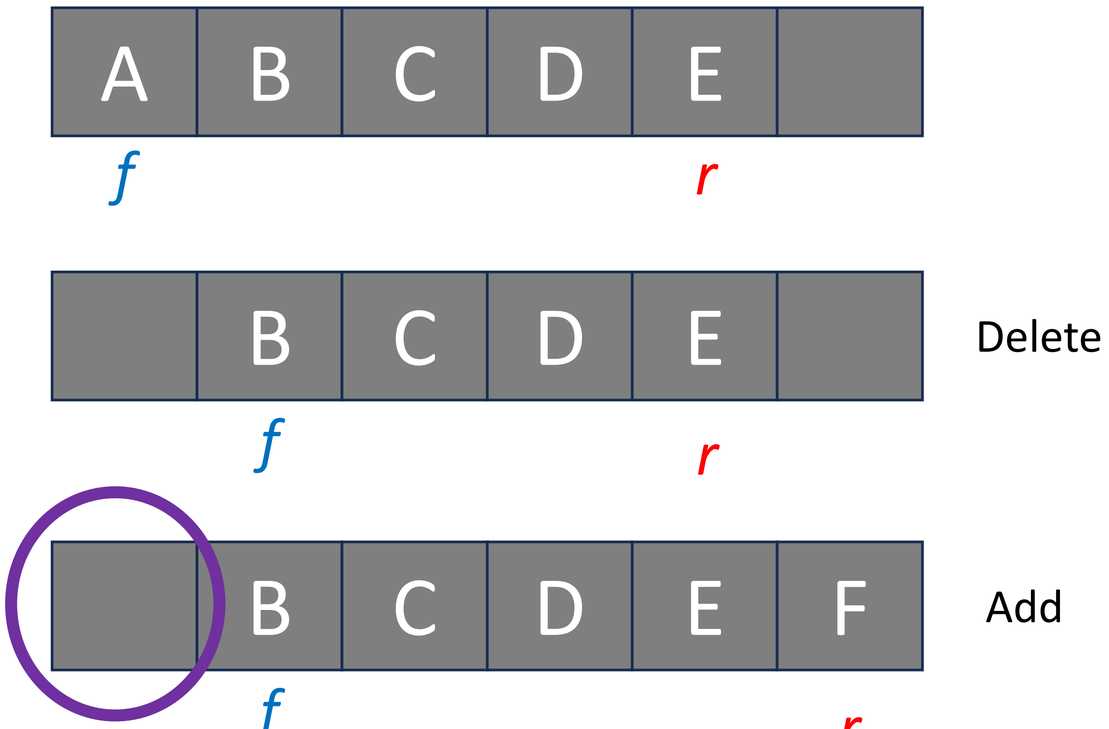
- **IsEmptyQ**: return True if queue is empty

  front == *rear*

- **AddQ**: insert item at rear of queue
- **DeleteQ**: remove and return the item at front of queue

Representation of queues (1)

Sequential representation



The queue is full because *r*== MAX_QUEUE_SIZE.

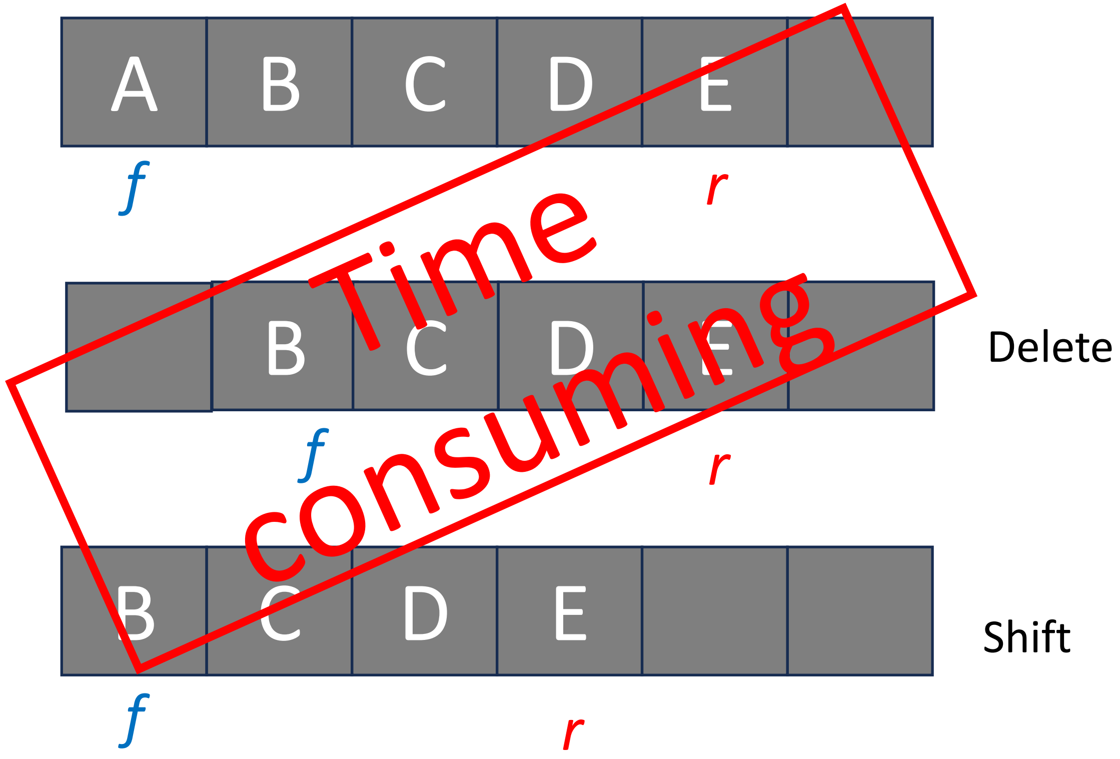
But the array is not full.

Some empty space remains.

When a customer at front of the line leaves, what will other customers do?

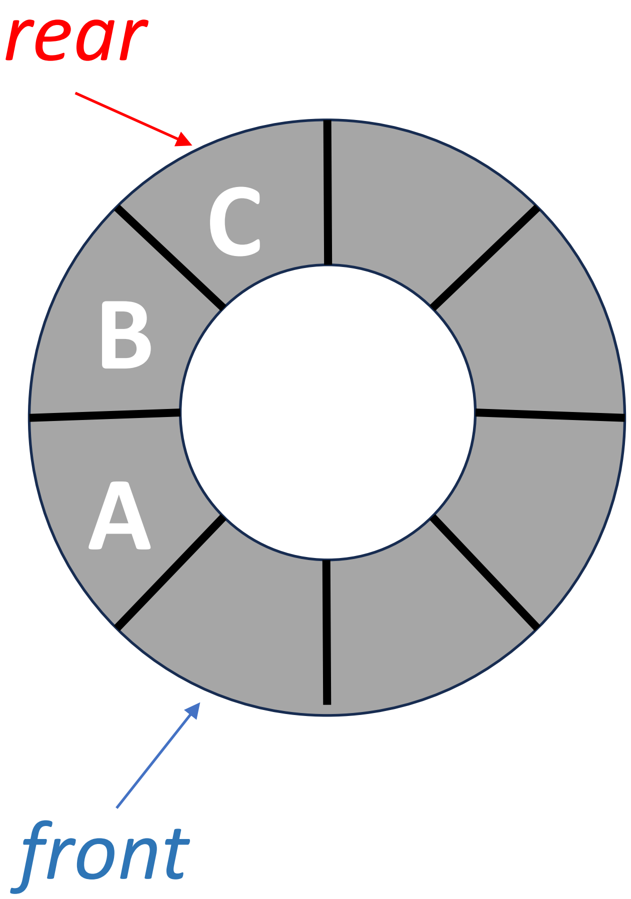
They move forward.

So, let's shit the array after each delete.



Representation of queues (2)

Circular queue



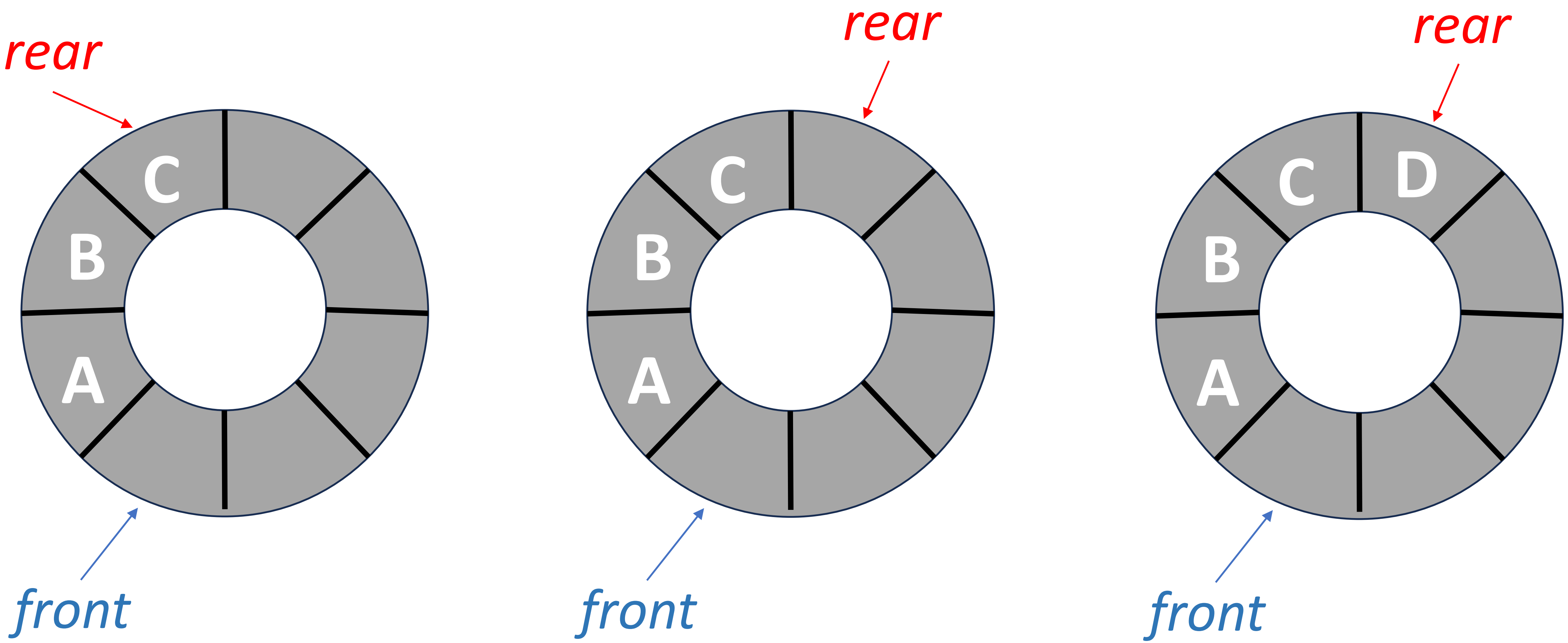
- *front*: One position counterclockwise from the position of the front element.
- *rear*: The position of the rear element.

Add an element into a circular queue

1. Move *rear* one clockwise.

```
rear = (rear+1) % MAX_QUEUE_SIZE
```

2. Put new element into queue[*rear*].

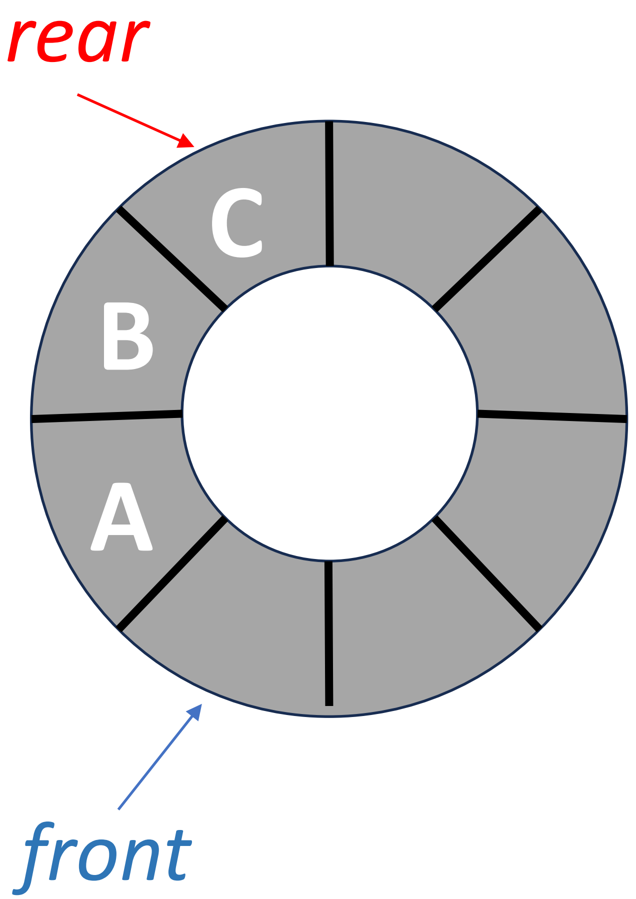


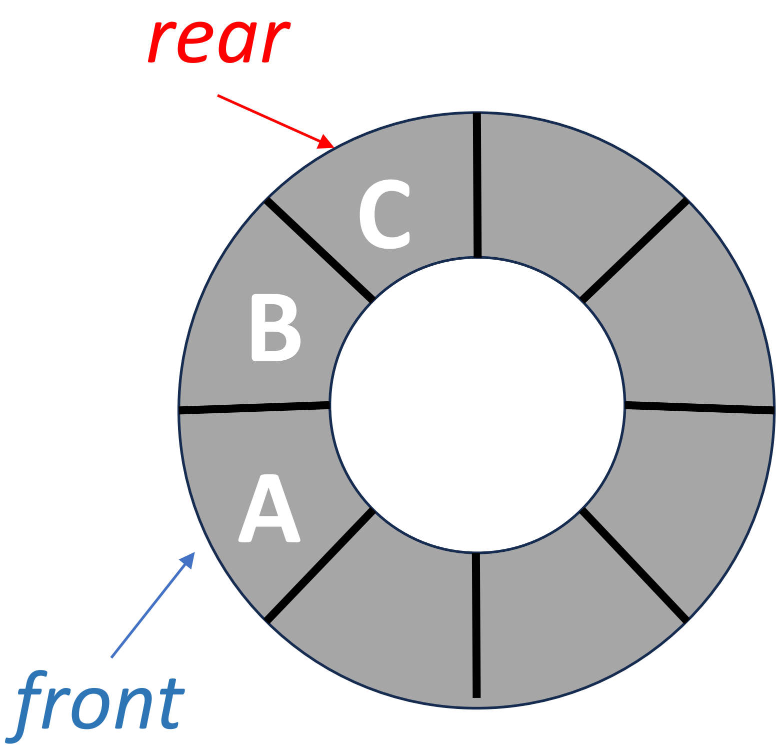
Delete an element from a circular queue

1. Move *front* one position clockwise.

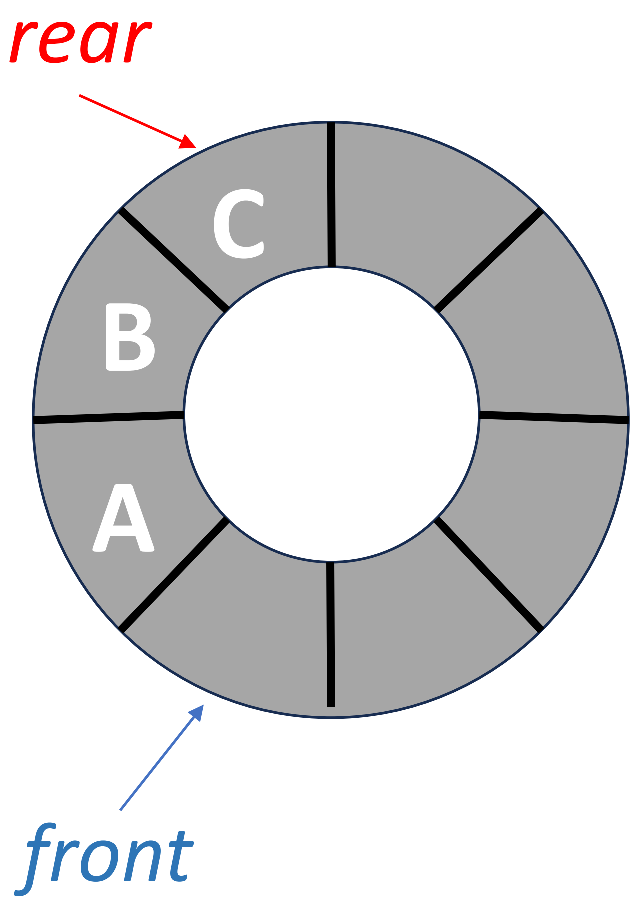
```
front = (front+1) % MAX_QUEUE_SIZE
```

2. Return value of queue[*front*].

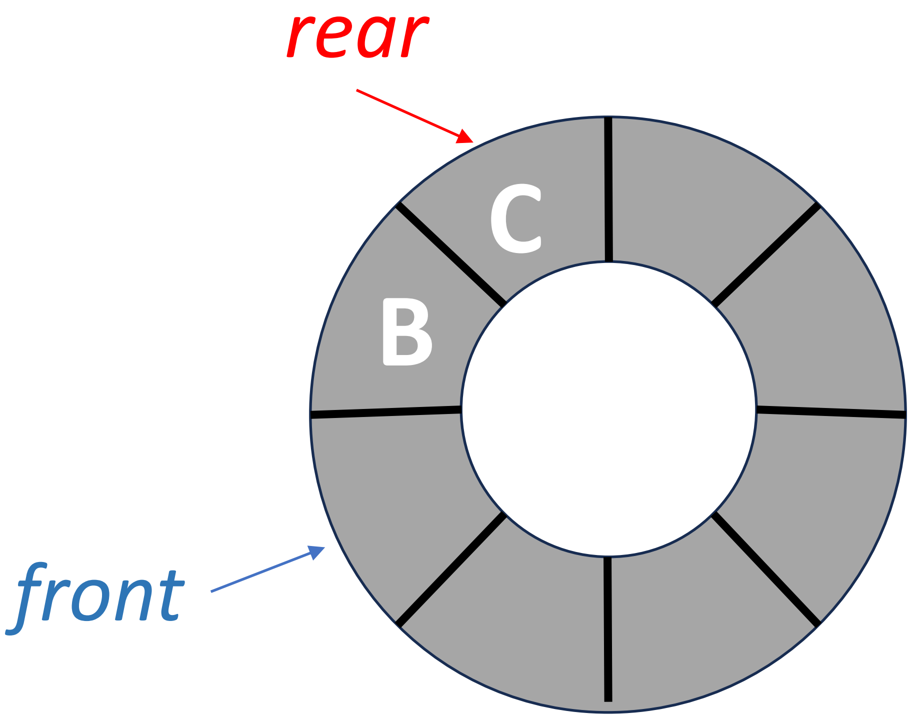




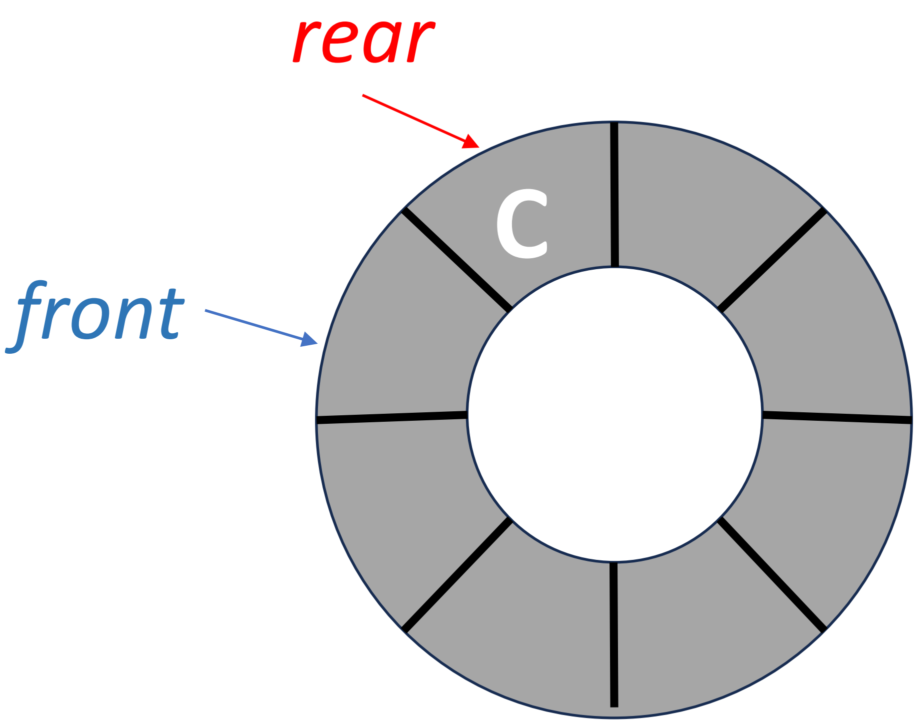
Emptying the circular queue



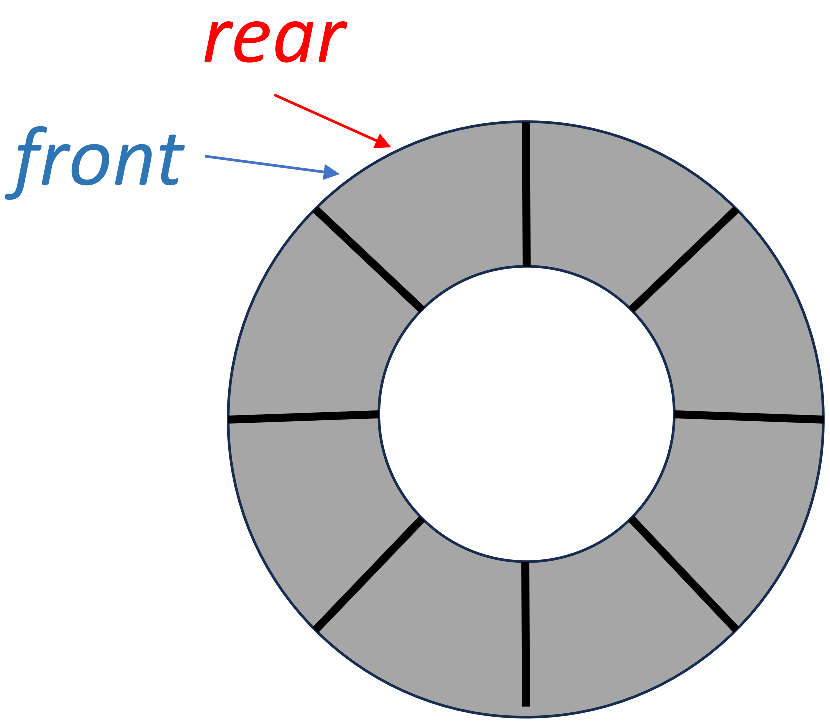
Emptying the circular queue



Emptying the circular queue

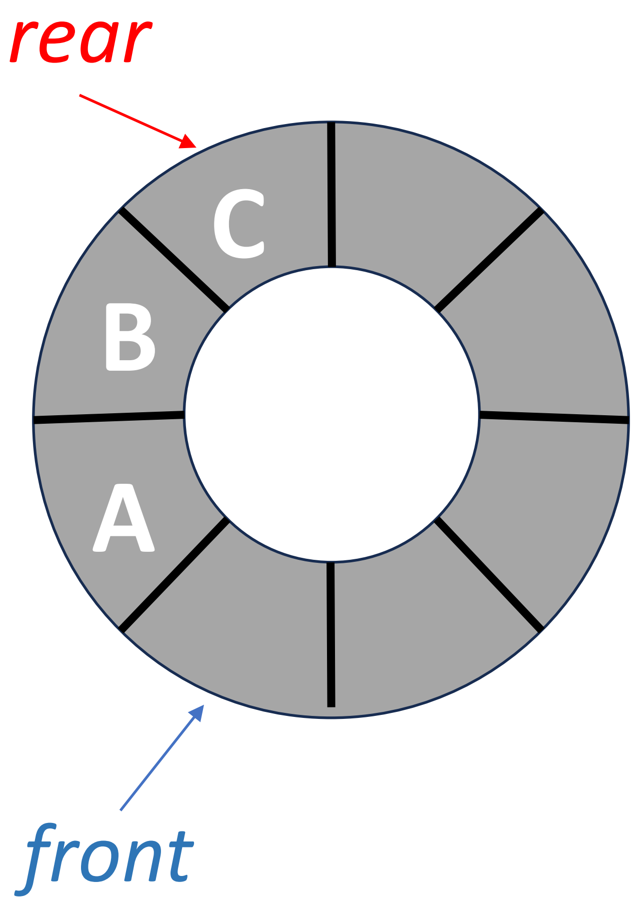


Emptying the circular queue

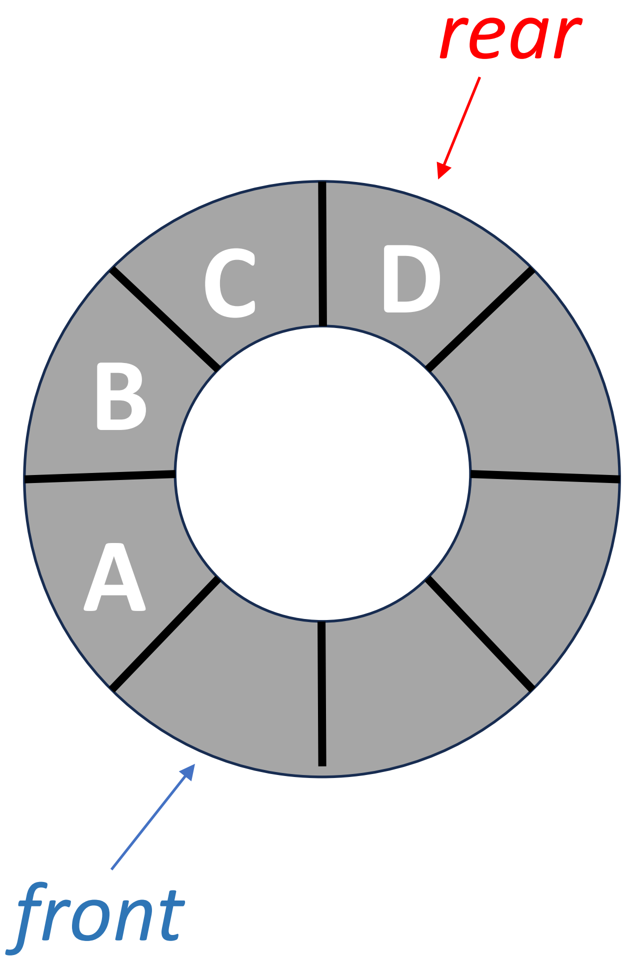


```
If (front==rear)
    return queueEmpty();
```

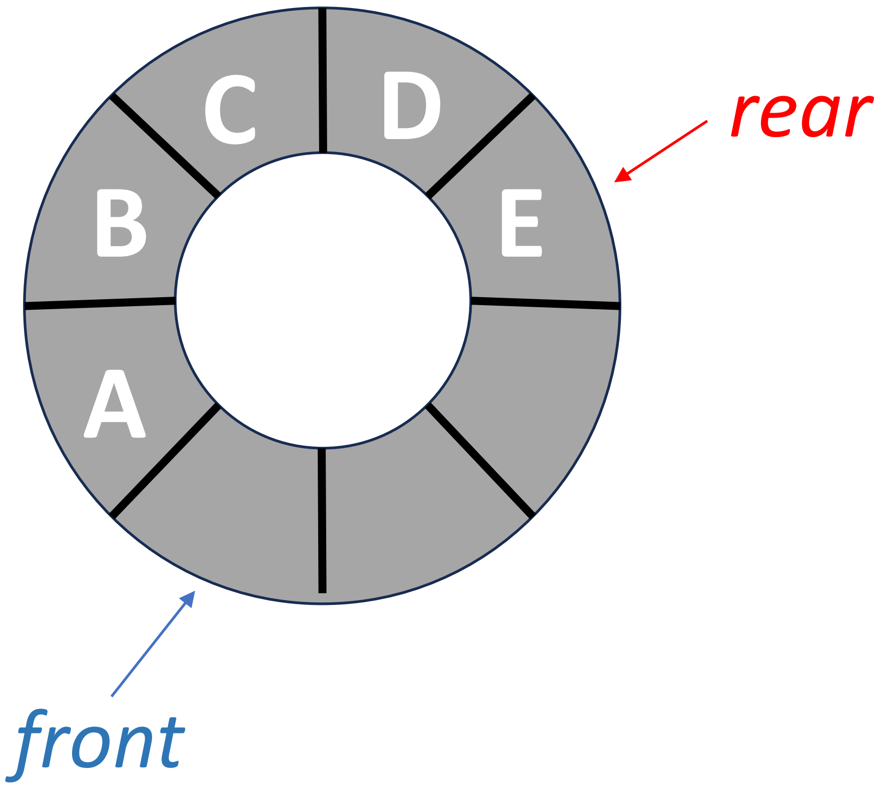
Addition until the circular queue is full



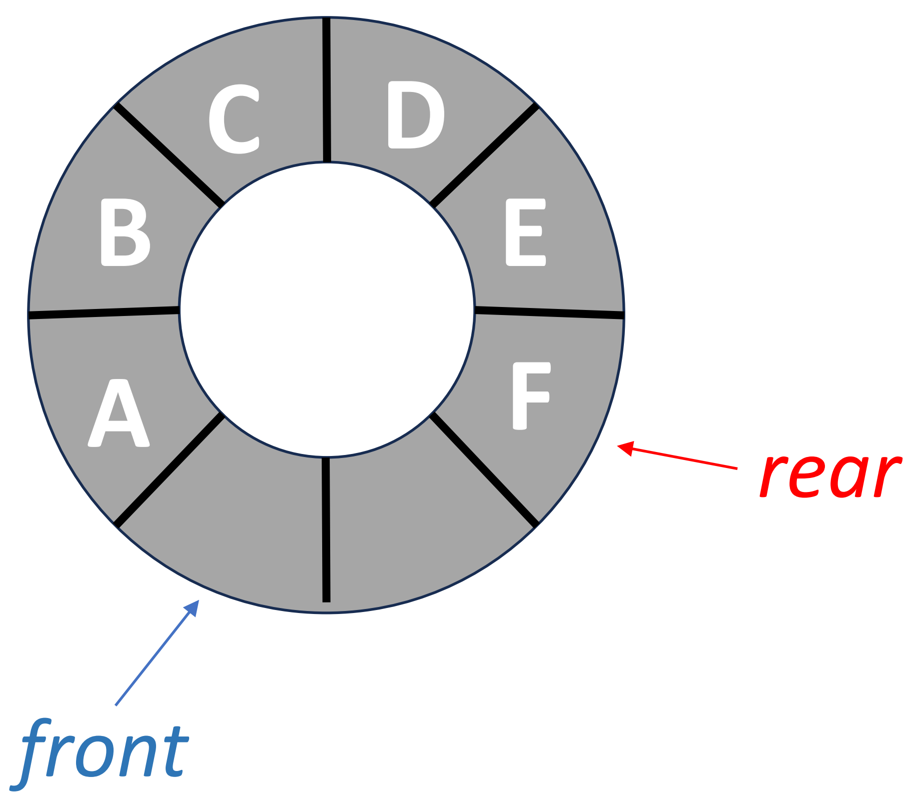
Addition until the circular queue is full



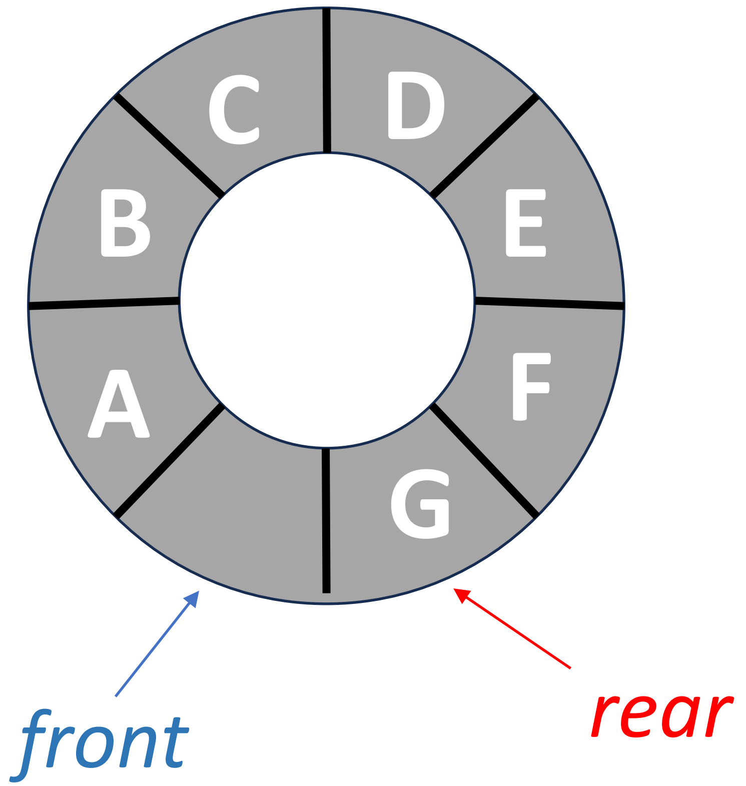
Addition until the circular queue is full



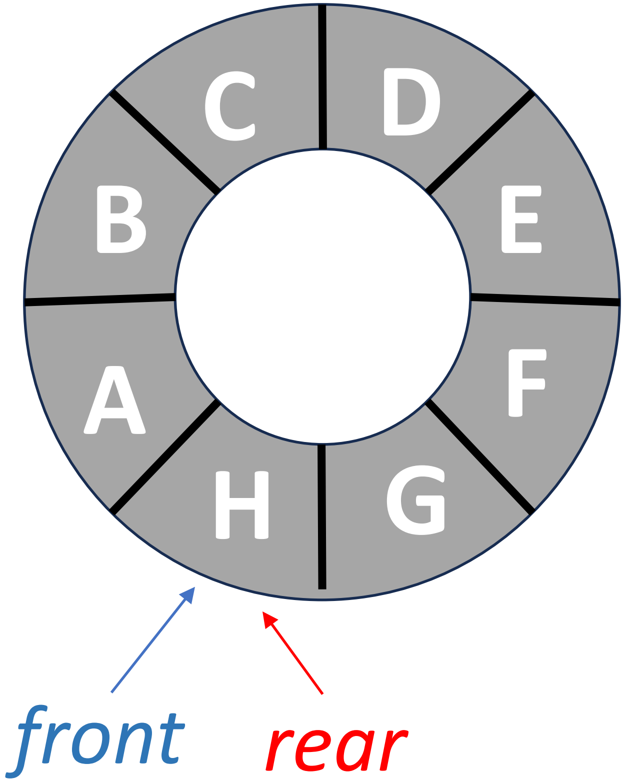
Addition until the circular queue is full



Addition until the circular queue is full



Addition until the circular queue is full



```
If (front==rear)
    return queueFull();
```

Edge conditions should be considered.

- In DeleteQ, test for an empty queue is *front* == *rear*.
- In AddQ, test for a full queue is also *front* == *rear*.

We cannot distinguish between the case of empty and full circular queues.

Q6: What can we do to avoid this?

Please reply your answers via the following link:

Group members: 1~3 people

Hint: No standard answer. There are several ways.

Possible solutions

1. Don’t let the queue get full.
   - When the addition of an element will signal *queueFull()*, increase array size.

2. Define a Boolean variable lastOperationIsAddQ.
   - Following each AddQ, set this variable to True.
   - Following each DeleteQ, set this variable to False.
   - Queue is empty if *front==rear* and !lastOperationIsAddQ
   - Queue is full if *front==rear* and lastOperationIsAddQ

Possible solutions

3. Define an integer variable size.
   - Following each AddQ do size++.
   - Following each DeleteQ do size--.
   - Queue is empty if (size == 0)
   - Queue is full if (size == arrayLength)