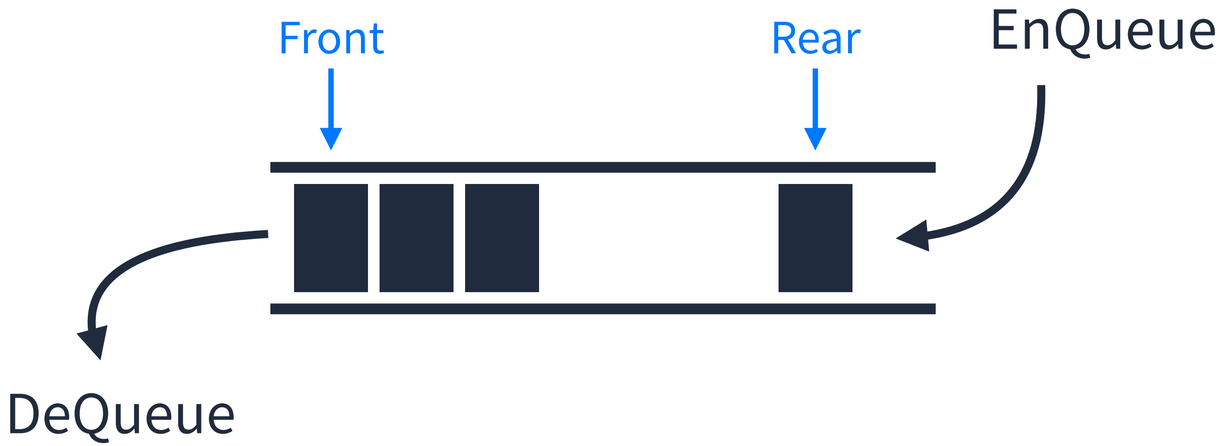
programmers

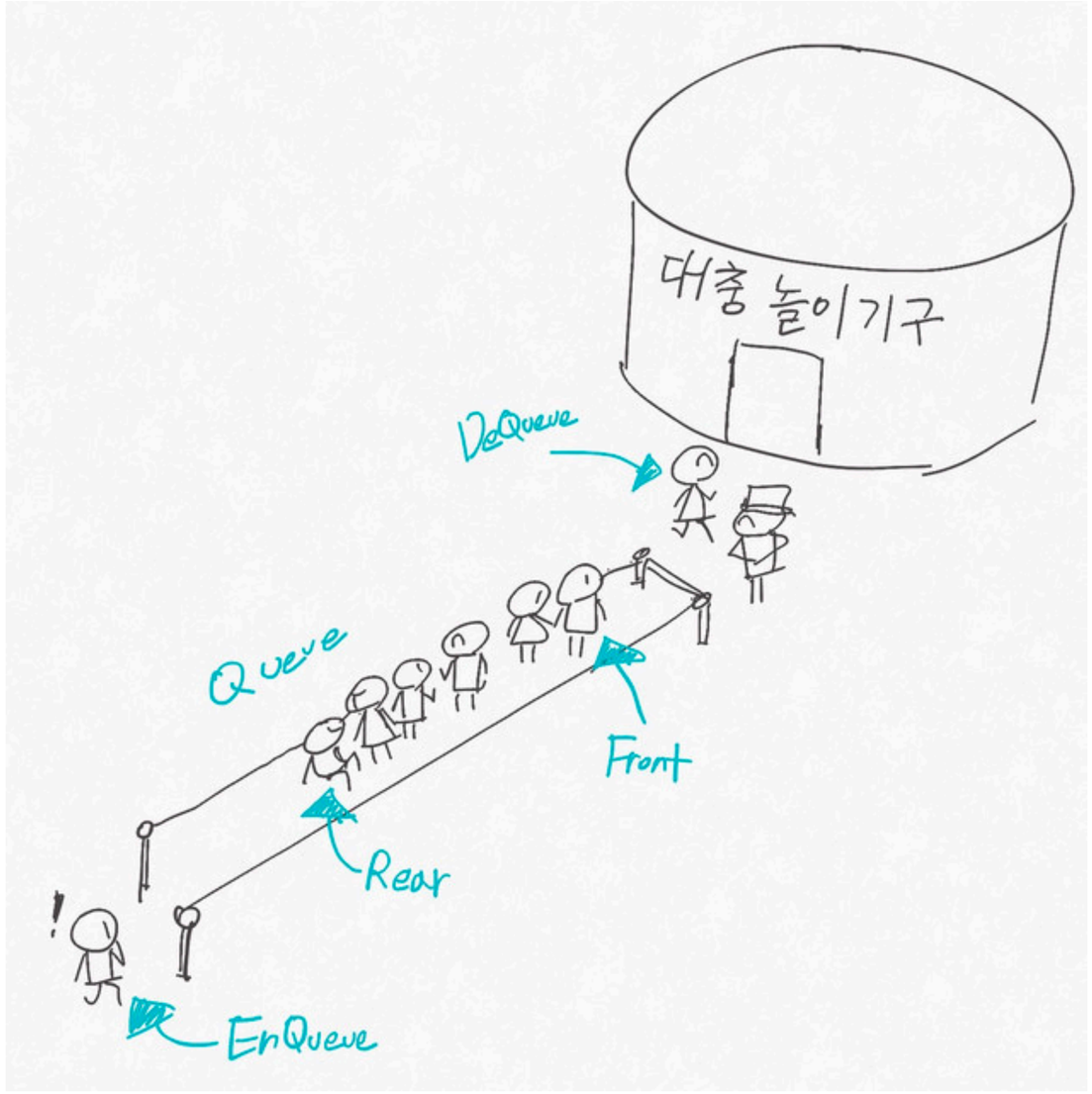
큐

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter

큐

First In First Out이라는 개념을 가진 선형 자료구조다.
Linear Queue와 Circular Queue가 존재한다.




대충 놀이기구
DeQueue
Queue
Front
Rear
EnQueue

Linear Queue

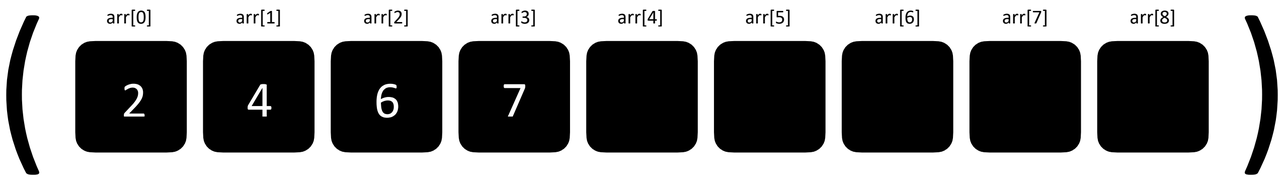
Array로 표현하기

Linear Queue를 Array로 표현할 수 있다.

Array로 표현하기

Linear Queue를 Array로 표현할 수 있다.

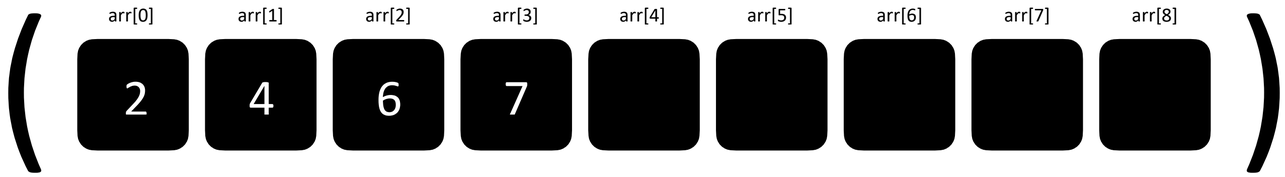
1



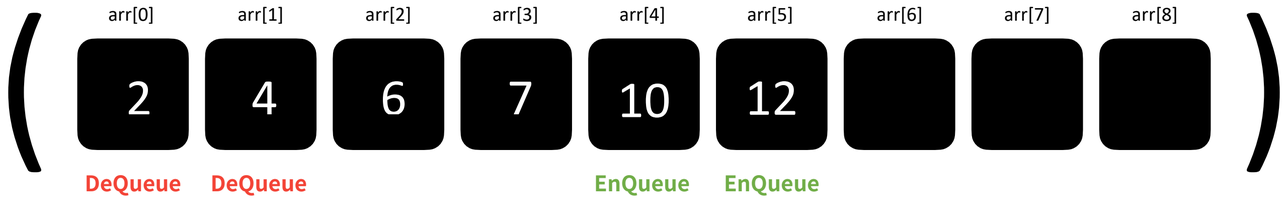
Array로 표현하기

Linear Queue를 Array로 표현할 수 있다.

1

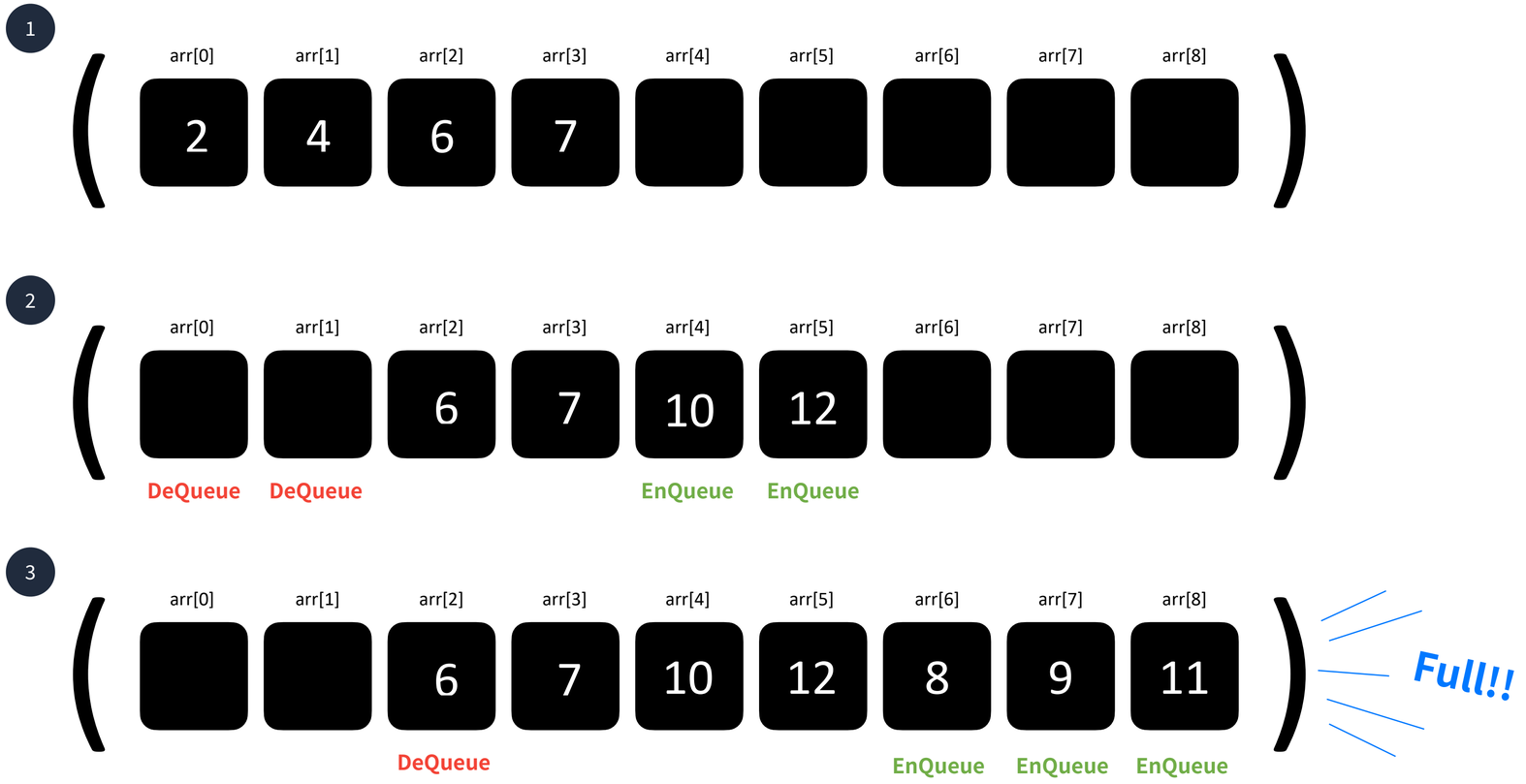


2



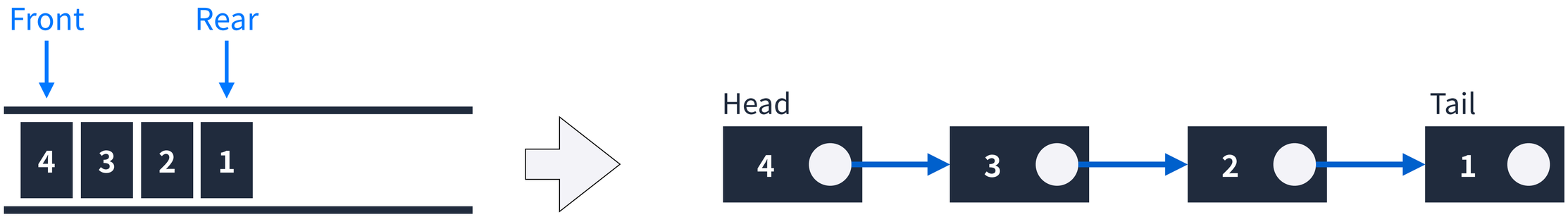
Array로 표현하기

Linear Queue를 Array로 표현할 수 있다.



Linked List로 표현하기

Linear Queue를 Linked List로 표현할 수 있다.



JavaScript에서 사용법

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Array로 구현

```
class Queue {
  constructor() {
    this.queue = [];
    this.front = 0;
    this.rear = 0;
  }

  enqueue(value) {
    this.queue[this.rear++] = value;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front += 1;
    return value;
  }

  peek() {
    return this.queue[this.front];
  }

  size() {
    return this.rear - this.front;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size()); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```js
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```js
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```js
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```js
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```js
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```js
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```js
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```js
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```js
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```js
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

Linked List로 구현

```
class Node {
  constructor(value) {
    this.value = value;
    this.next = null;
  }
}

class Queue {
  constructor() {
    this.head = null;
    this.tail = null;
    this.size = 0;
  }

  enqueue(newValue) {
    const newNode = new Node(newValue);
    if (this.head === null) {
      this.head = this.tail = newNode;
    } else {
      this.tail.next = newNode;
      this.tail = newNode;
    }
    this.size += 1;
  }

  dequeue() {
    const value = this.head.value;
    this.head = this.head.next;
    this.size -= 1;
    return value;
  }

  peek() {
    return this.head.value;
  }
}
```

```
const queue = new Queue();
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
console.log(queue.dequeue()); // 1
queue.enqueue(8);
console.log(queue.size); // 3
console.log(queue.peek()); // 2
console.log(queue.dequeue()); // 2
console.log(queue.dequeue()); // 4
```

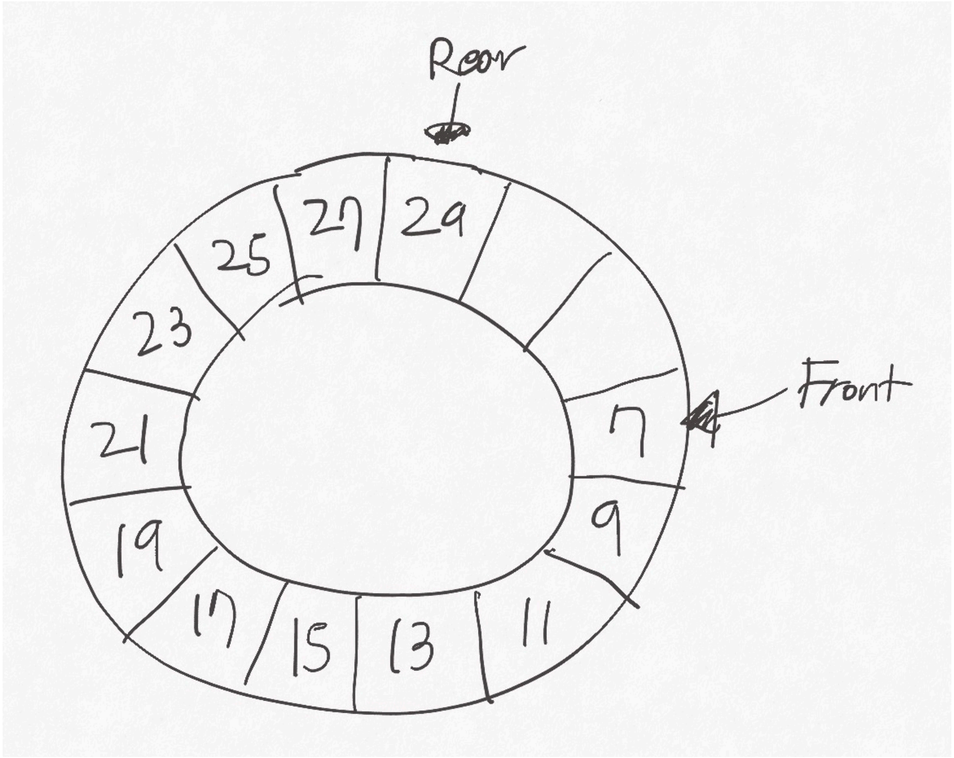
shift 함수는 쓰지 마세요!

```
const queue = [1, 2, 3];
queue.push(4);
const value = queue.shift(); // O(n) !!
console.log(value); // 1
```

Circular Queue

Circular Queue

Front와 Rear가 이어져있는 Queue
Circular Queue는 Linked List로 구현했을 때 이점이 없다.



Array로 구현

```
class Queue {
  constructor(maxSize) {
    this.maxSize = maxSize;
    this.queue = [];
    this.front = 0;
    this.rear = 0;
    this.size = 0;
  }

  enqueue(value) {
    if (this.isFull()) {
      console.log("Queue is full.");
      return;
    }
    this.queue[this.rear] = value;
    this.rear = (this.rear + 1) % this.maxSize;
    this.size += 1;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front = (this.front + 1) % this.maxSize;
    this.size -= 1;
    return value;
  }

  isFull() {
    return this.size === this.maxSize;
  }

  peek() {
    return this.queue[this.front];
  }
}
```

```
const queue = new Queue(4);
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
queue.enqueue(8);
queue.enqueue(16); // Queue is full.
console.log(queue.dequeue()); // 1
console.log(queue.dequeue()); // 2
console.log(queue.size); // 2
console.log(queue.peek()); // 4
queue.enqueue(16);
queue.enqueue(32);
console.log(queue.isFull()); // true
```

Array로 구현

```
class Queue {
  constructor(maxSize) {
    this.maxSize = maxSize;
    this.queue = [];
    this.front = 0;
    this.rear = 0;
    this.size = 0;
  }

  enqueue(value) {
    if (this.isFull()) {
      console.log("Queue is full.");
      return;
    }
    this.queue[this.rear] = value;
    this.rear = (this.rear + 1) % this.maxSize;
    this.size += 1;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front = (this.front + 1) % this.maxSize;
    this.size -= 1;
    return value;
  }

  isFull() {
    return this.size === this.maxSize;
  }

  peek() {
    return this.queue[this.front];
  }
}
```

```
const queue = new Queue(4);
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
queue.enqueue(8);
queue.enqueue(16); // Queue is full.
console.log(queue.dequeue()); // 1
console.log(queue.dequeue()); // 2
console.log(queue.size); // 2
console.log(queue.peek()); // 4
queue.enqueue(16);
queue.enqueue(32);
console.log(queue.isFull()); // true
```

Array로 구현

```
class Queue {
  constructor(maxSize) {
    this.maxSize = maxSize;
    this.queue = [];
    this.front = 0;
    this.rear = 0;
    this.size = 0;
  }

  enqueue(value) {
    if (this.isFull()) {
      console.log("Queue is full.");
      return;
    }
    this.queue[this.rear] = value;
    this.rear = (this.rear + 1) % this.maxSize;
    this.size += 1;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front = (this.front + 1) % this.maxSize;
    this.size -= 1;
    return value;
  }

  isFull() {
    return this.size === this.maxSize;
  }

  peek() {
    return this.queue[this.front];
  }
}
```

```
const queue = new Queue(4);
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
queue.enqueue(8);
queue.enqueue(16); // Queue is full.
console.log(queue.dequeue()); // 1
console.log(queue.dequeue()); // 2
console.log(queue.size); // 2
console.log(queue.peek()); // 4
queue.enqueue(16);
queue.enqueue(32);
console.log(queue.isFull()); // true
```

Array로 구현

```
class Queue {
  constructor(maxSize) {
    this.maxSize = maxSize;
    this.queue = [];
    this.front = 0;
    this.rear = 0;
    this.size = 0;
  }

  enqueue(value) {
    if (this.isFull()) {
      console.log("Queue is full.");
      return;
    }
    this.queue[this.rear] = value;
    this.rear = (this.rear + 1) % this.maxSize;
    this.size += 1;
  }

  dequeue() {
    const value = this.queue[this.front];
    delete this.queue[this.front];
    this.front = (this.front + 1) % this.maxSize;
    this.size -= 1;
    return value;
  }

  isFull() {
    return this.size === this.maxSize;
  }

  peek() {
    return this.queue[this.front];
  }
}
```

```
const queue = new Queue(4);
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(4);
queue.enqueue(8);
queue.enqueue(16); // Queue is full.
console.log(queue.dequeue()); // 1
console.log(queue.dequeue()); // 2
console.log(queue.size); // 2
console.log(queue.peek()); // 4
queue.enqueue(16);
queue.enqueue(32);
console.log(queue.isFull()); // true
```

programmers

큐

코딩테스트 광탈방지 A to Z : JavaScript - 이선협 @kciter