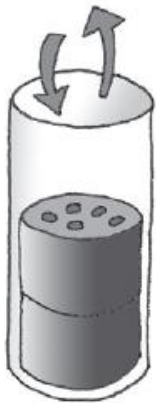


5 큐

1. 큐의 이해 : 큐의 개념과 구조

❖ 큐(Queue)

- 스택과 비슷한 삽입과 삭제의 위치가 제한되어있는 유한 순서 리스트
 - 큐는 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 구조
 - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(**F**irst-**I**n)한 원소는 맨 앞에 있다가 가장 먼저 삭제(**F**irst-**O**ut)됨
- ☞ **선입선출 구조 (FIFO, First-In-First-Out)**



(a) 스택의 구조



(b) 큐의 구조

그림 6-1 스택과 큐의 구조 비교 예

1. 큐의 이해 : 큐의 개념과 구조

■ FIFO 구조의 예

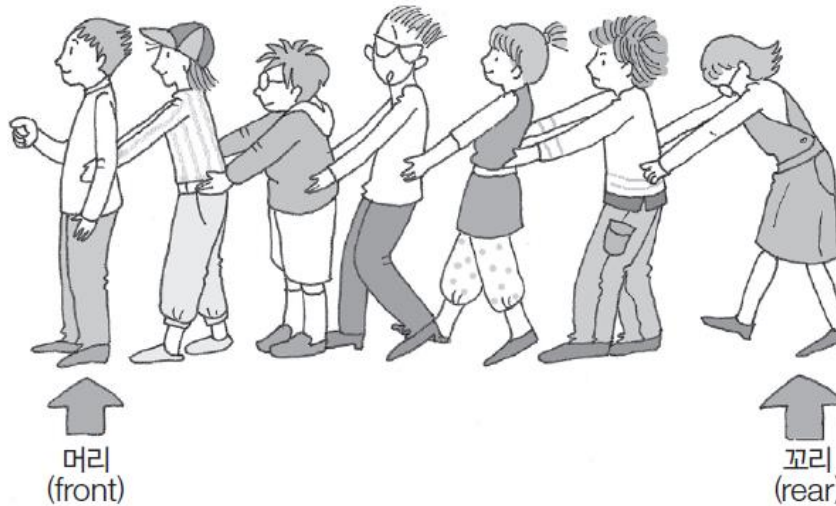


그림 6-2 FIFO 구조의 예 : 꼬리잡기 놀이의 머리와 꼬리

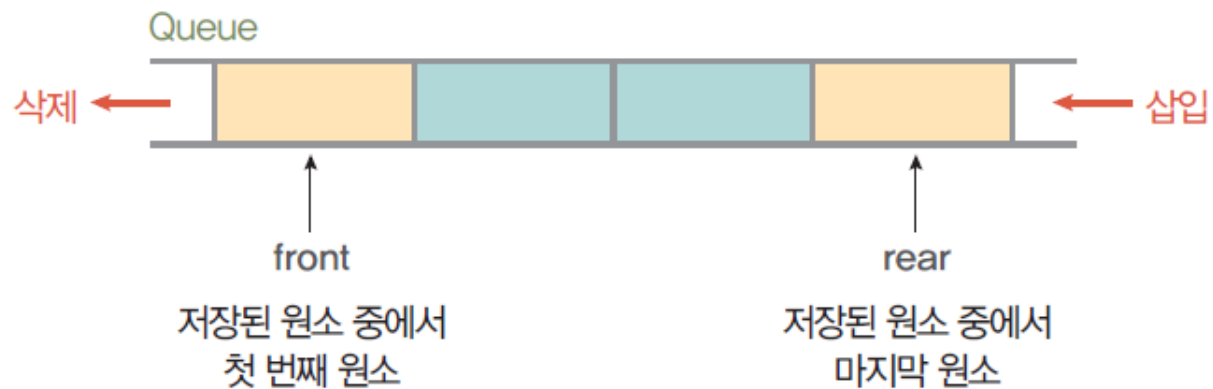


그림 6-3 큐의 FIFO 구조



1. 큐의 이해 : 큐의 개념과 구조

- 큐의 연산
 - 삽입 : **enQueue**
 - 삭제 : **deQueue**
- 스택과 큐의 연산 비교

표 6-1 스택과 큐에서의 삽입과 삭제 연산 비교

자료구조 \ 항목	삽입 연산		삭제 연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	push	top	pop	top
큐	enQueue	rear	deQueue	front



1. 큐의 이해 : 큐의 추상 자료형

ADT 6-1

큐의 추상 자료형

ADT Queue

데이터 : 0개 이상의 원소를 가진 유한 순서 리스트

연산 : $Q \in \text{Queue}$; $\text{item} \in \text{Element}$;

// 공백 큐를 생성하는 연산

`createQueue() ::= create an empty Q;`

// 큐가 공백 상태인지 검사하는 연산

`isEmpty(Q) ::= if (Q is empty) then return true
 else return false;`

// 큐의 rear에 원소를 삽입하는 연산

`enqueue(Q, item) ::= insert item at the rear of Q;`

// 큐의 front에 있는 원소를 삭제하는 연산

`dequeue(Q) ::= if (isEmpty(Q)) then return error
 else { delete and return the front item Q };`

// 큐의 front에 있는 원소를 반환하는 연산

`peek(Q) ::= if (isEmpty(Q)) then return error
 else { return the front item of the Q };`

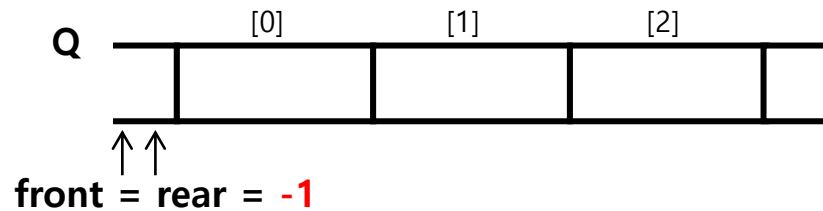
End Queue



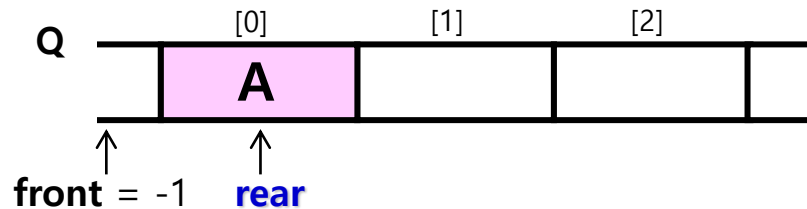
1. 큐의 이해

■ 큐의 연산 과정

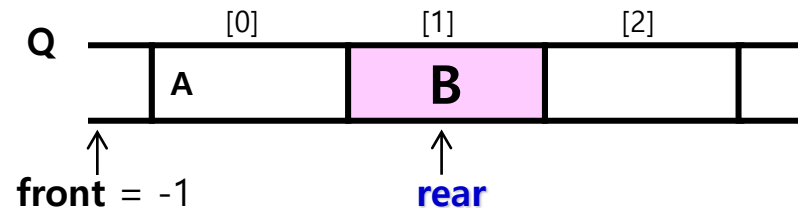
- ① 공백 큐 생성 : `createQueue();`



- ② 원소 A 삽입 : `enqueue(Q, A);`

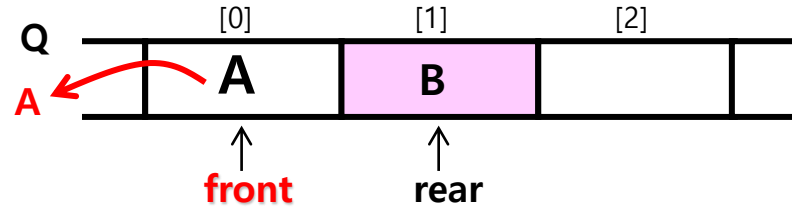


- ③ 원소 B 삽입 : `enqueue(Q, B);`

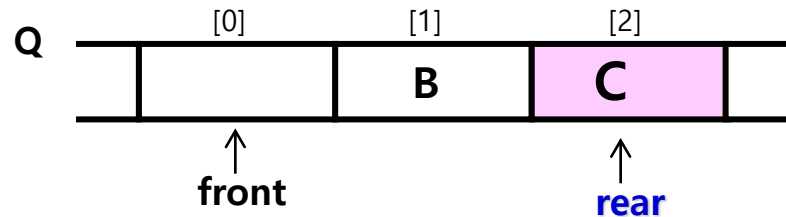


1. 큐

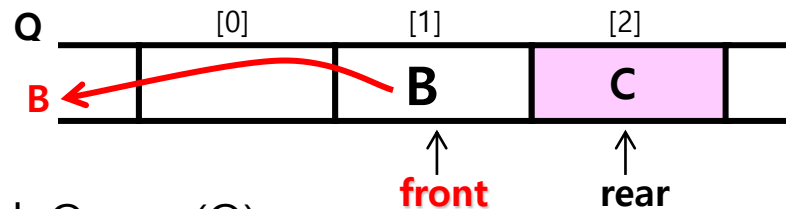
- ④ 원소 삭제 : `deQueue(Q);`



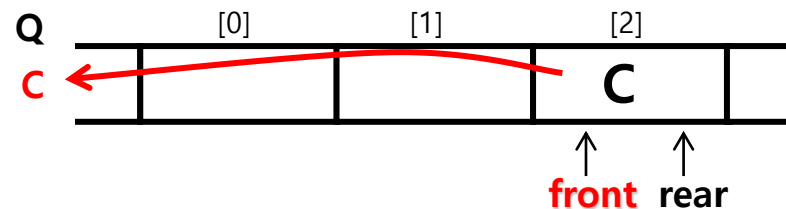
- ⑤ 원소 `C` 삽입 : `enqueue(Q, C);`



- ⑥ 원소 삭제 : `deQueue(Q);`



- ⑦ 원소 삭제 : `deQueue(Q);`



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

❖ 순차 큐

■ 1차원 배열을 이용한 큐

- 큐의 크기 = 배열의 크기
- 변수 front : 저장된 첫 번째 원소의 인덱스 저장
- 변수 rear : 저장된 마지막 원소의 인덱스 저장

■ 상태 표현

- 초기 상태 : $\text{front} = \text{rear} = -1$
- 공백 상태 : $\text{front} = \text{rear}$
- 포화 상태 : $\text{rear} = n-1$ (n : 배열의 크기, $n-1$: 배열의 마지막 인덱스)



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

- 초기 공백 큐 생성 알고리즘
 - 크기가 n 인 1차원 배열 생성
 - front와 rear를 -1로 초기화

알고리즘 6-1 공백 순차 큐 생성

```
createQueue()  
  Q[n];  
  front  $\leftarrow$  -1;  
  rear  $\leftarrow$  -1;  
end createQueue()
```



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

- 공백 큐 검사 알고리즘과 포화상태 검사 알고리즘
 - 공백 상태 : $\text{front} = \text{rear}$
 - 포화 상태 : $\text{rear} = n-1$ (n : 배열의 크기, $n-1$: 배열의 마지막 인덱스)

알고리즘 6-2 순차큐 공백 상태 검사

```
isEmpty(Q)
  if (front = rear) then return true;
  else return false;
end isEmpty()
```

알고리즘 6-3 순차큐의 포화 상태 검사

```
isFull(Q)
  if (rear = n - 1) then return true;
  else return false;
end isFull()
```



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

■ 큐의 삽입 알고리즘

알고리즘 6-4 순차큐의 원소 삽입

```
enqueue(Q, item)
  if (isFull(Q)) then Queue_Full();    // 포화 상태이면 삽입 연산 중단
  else {
    ❶ rear ← rear + 1;
    ❷ Q[rear] ← item;
  }
end enqueue()
```

- 마지막 원소의 뒤에 삽입해야 하므로

- ❶ 마지막 원소의 인덱스를 저장한 **rear**의 값을 하나 증가시켜 삽입할 자리 준비
- ❷ 수정한 rear값에 해당하는 배열원소 Q[rear]에 item을 저장



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

■ 큐의 삭제 알고리즘

알고리즘 6-5 순차 큐의 원소 삭제

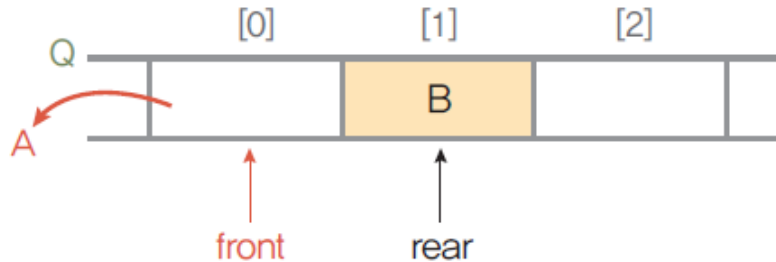
```
deQueue(Q)
  if (isEmpty(Q)) then Queue_Empty();    // 공백 상태이면 삭제 연산 중단
  else {
    ❶ front ← front + 1;
    ❷ return Q[front];
  }
end deQueue()
```

- 가장 앞에 있는 원소를 삭제해야 하므로

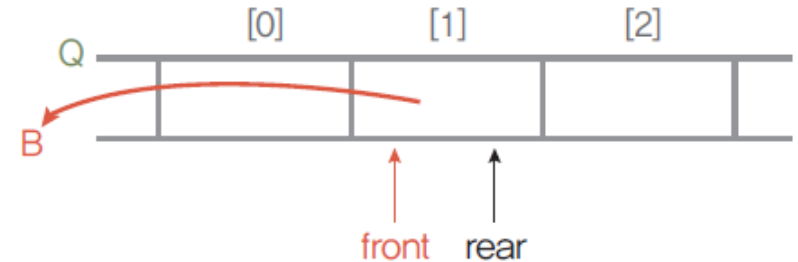
- ❶ **front**의 위치를 한자리 뒤로 이동하여 큐에 남아있는 첫 번째 원소의 위치로 이동하여 삭제할 자리 준비
- ❷ front 자리의 원소를 삭제하여 반환



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현



(a) 첫 번째 `deQueue()` 연산 후 상태



(b) 두 번째 `deQueue()` 연산 후 상태

그림 6-4 `deQueue()` 연산 후 상태



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

■ 큐의 검색 알고리즘

알고리즘 6-6 순차 큐의 원소 검색

```
peek(Q)
  if (isEmpty(Q)) then Queue_Empty();
  else return Q[front + 1];
end peek()
```

- 가장 앞에 있는 원소를 검색하여 반환하는 연산

① 현재 **front**의 한자리 뒤(front+1)에 있는 원소, 즉 큐에 있는 첫 번째 원소를 반환



2. 큐의 구현 : 순차자료구조를 이용한 큐의 구현

- 순차 자료구조를 이용해 순차 큐 구현하기 프로그램 : [교재 280p](#)
- 실행 결과

```
CA 명령 프롬프트

***** 순차 큐 연산 *****

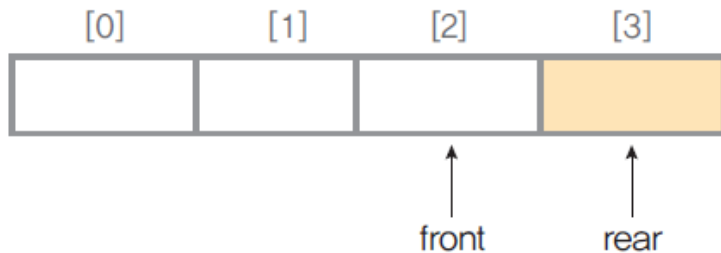
삽입 A>> Queue : [ A ]
삽입 B>> Queue : [ A B ]
삽입 C>> Queue : [ A B C ] peek item : A

삭제 >> Queue : [ B C ]   삭제 데이터 : A
삭제 >> Queue : [ C ]     삭제 데이터 : B
삭제 >> Queue : [ ]       삭제 데이터 : C
삽입 D>> Queue : [ D ]
삽입 E>> Queue is full! Queue : [ D ]
```

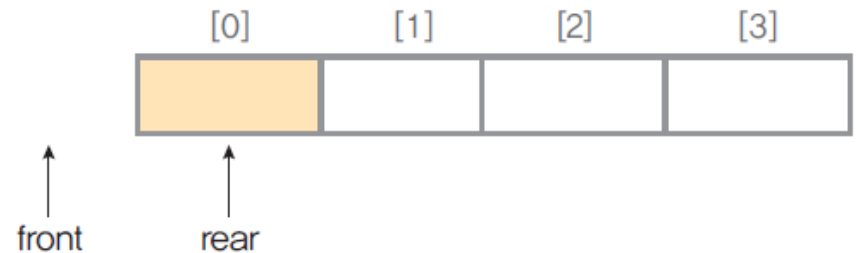


2. 큐의 구현 : 원형 큐의 구현

- 순차 큐의 잘못된 포화상태 인식
 - 큐에서 삽입과 삭제를 반복하면서 그림(a)와 같은 상태일 경우, 앞부분에 빈자리가 있지만 $rear=n-1$ 상태이므로 포화상태로 인식하고 더 이상의 삽입을 수행하지 않는다.
- 순차 큐의 잘못된 포화상태 인식의 해결 방법-1
 - 저장된 원소들을 배열의 앞부분으로 이동시키기
 - 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐



(a) 포화 상태로 잘못 인식하는 경우



(b) 큐의 원소들을 앞으로 이동하여 해결

그림 6-5 순차 큐의 잘못된 포화 상태 문제와 해결 방법



2. 큐의 구현 : 원형 큐의 구현

- 순차 큐의 잘못된 포화상태 인식의 해결 방법-2
 - 1차원 배열을 사용하면서 논리적으로 배열의 처음과 끝이 연결되어 있다고 가정하고 사용 \Rightarrow 원형 큐
 - 원형 큐의 논리적 구조

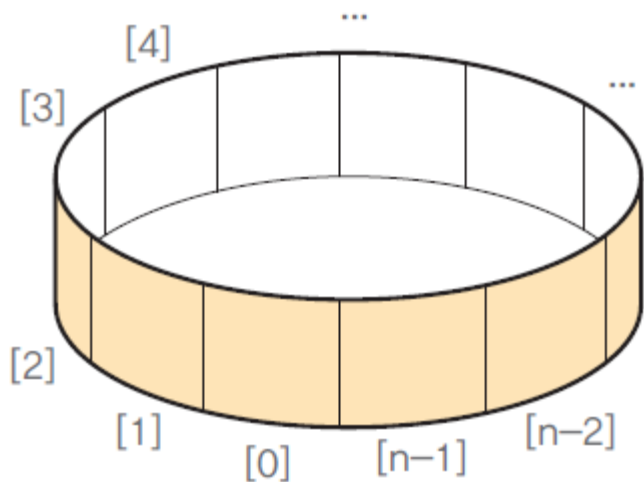


그림 6-6 원형 큐의 논리적 구조



2. 큐의 구현 : 원형 큐의 구현

■ 원형 큐의 구조

- 초기 공백 상태 : $\text{front} = \text{rear} = 0$
- front 와 rear 의 위치가 배열의 마지막 인덱스 $n-1$ 에서 논리적인 다음 자리인 인덱스 0번으로 이동하기 위해서 **나머지연산자 mod**를 사용
 - $3 \div 4 = 0 \dots 3$ (몫=0, 나머지=3)
 - $3 \bmod 4 = 3$

표 6-2 순차 큐와 원형 큐의 비교

종류	삽입 위치	삭제 위치
순차 큐	$\text{rear} = \text{rear} + 1$	$\text{front} = \text{front} + 1$
원형 큐	$\text{rear} = (\text{rear} + 1) \bmod n$	$\text{front} = (\text{front} + 1) \bmod n$

- 사용조건) 공백 상태와 포화 상태 구분을 쉽게 하기 위해서 front 가 있는 자리는 사용하지 않고 항상 빈자리로 둬



2. 큐의 구현 : 원형 큐의 구현

- 초기 공백 원형 큐 생성 알고리즘
 - 크기가 n 인 1차원 배열 생성
 - front와 rear를 0으로 초기화

알고리즘 6-7 공백 원형 큐 생성

```
createQueue()  
  cQ[n];  
  front ← 0;  
  rear ← 0;  
end createQueue()
```



2. 큐의 구현 : 원형 큐의 구현

- 원형 큐의 공백상태 검사 알고리즘과 포화상태 검사 알고리즘

알고리즘 6-8 원형 큐의 공백 상태 검사

```
isEmpty(cQ)
  if (front = rear) then return true;
  else return false;
end isEmpty()
```

알고리즘 6-9 원형 큐의 포화 상태 검사

```
isFull(cQ)
  if (((rear + 1) mod n) = front) then return true;
  else return false;
end isFull()
```

표 6-3 원형 큐의 상태에 따른 front와 rear의 관계

구분	조건
공백 상태	front = rear
포화 상태	$(rear+1) \bmod n = front$



2. 큐의 구현 : 원형 큐의 구현

■ 원형 큐의 삽입 알고리즘

- ① rear의 값을 조정하여 삽입할 자리를 준비 : $\text{rear} \leftarrow (\text{rear} + 1) \bmod n$;
- ② 준비한 자리 $\text{cQ}[\text{rear}]$ 에 원소 item 을 삽입

알고리즘 6-10 원형 큐의 원소 삽입

```
enqueue(cQ, item)
  if (isFull(cQ)) then Queue_Full(); // 포화 상태이면 삽입 연산 중단
  else {
    ① rear  $\leftarrow$  (rear + 1) mod n;
    ② cQ[rear]  $\leftarrow$  item;
  }
end enqueue()
```



2. 큐의 구현 : 원형 큐의 구현

■ 원형 큐의 삭제 알고리즘

- ① front의 값을 조정하여 삭제할 자리를 준비
- ② 준비한 자리에 있는 원소 cQ[front]를 삭제하여 반환

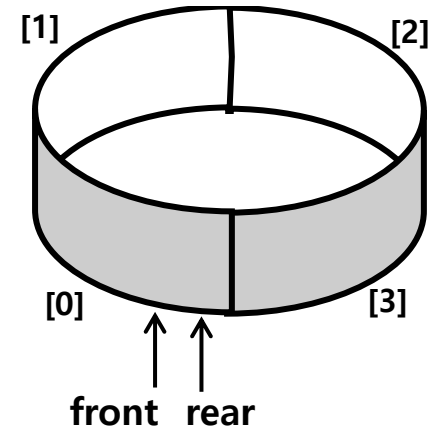
알고리즘 6-11 원형 큐의 원소 삭제

```
deQueue(cQ)
  if (isEmpty(cQ)) then Queue_Empty(); // 공백 상태이면 삭제 연산 중단
  else {
    ① front ← (front + 1) mod n;
    ② return cQ[front];
  }
end deQueue()
```

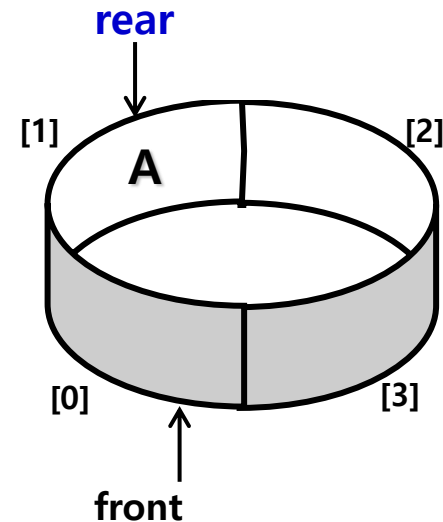


2. 큐의 구현 : 원형 큐의 구현

- 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정
- ① 공백 원형 큐 생성 : `createQueue()`;

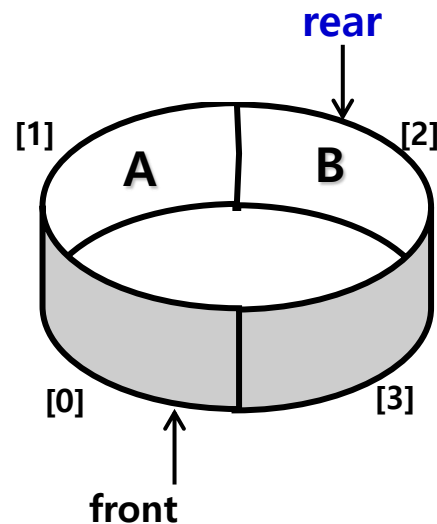


- ② 원소 A 삽입 : `enQueue(cQ, A)`;

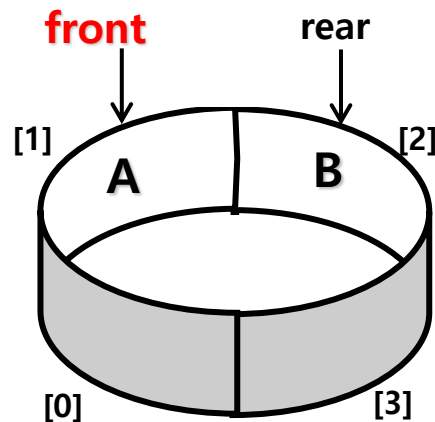


2. 큐의 구현 : 원형 큐의 구현

③ 원소 B 삽입 : `enqueue(cQ, B);`

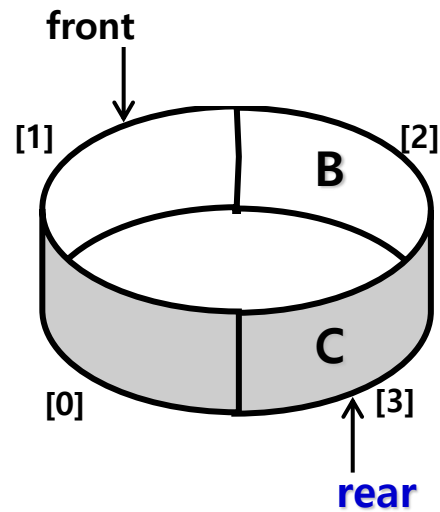


④ 원소 삭제 : `dequeue(cQ);`
(삭제 데이터 : A)

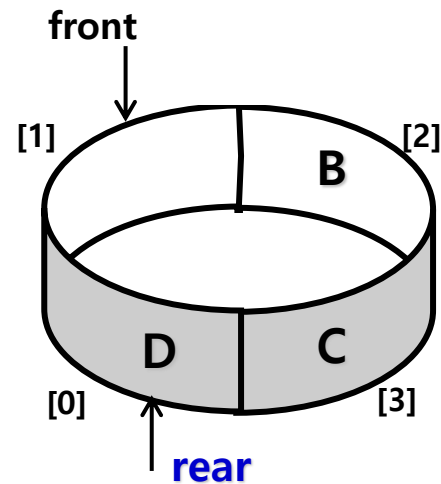


2. 큐의 구현 : 원형 큐의 구현

⑤ 원소 C 삽입 : `enqueue(cQ, C);`



⑥ 원소 D 삽입 : `enqueue(cQ, D);`



2. 큐의 구현 : 원형 큐의 구현

- 순차 자료구조를 이용해 원형 큐 구현하기 프로그램 : [교재 287p](#)
- 실행 결과

```
cmd 명령 프롬프트

***** 원형 큐 연산 *****

삽입 A>> Circular Queue : [ A ]
삽입 B>> Circular Queue : [ A B ]
삽입 C>> Circular Queue : [ A B C ] peek item : A

삭제 >> Circular Queue : [ B C ] 삭제 데이터 : A
삭제 >> Circular Queue : [ C ]   삭제 데이터 : B
삭제 >> Circular Queue : [ ]     삭제 데이터 : C
삽입 D>> Circular Queue : [ D ]
삽입 E>> Circular Queue : [ D E ]
```





Thank You

