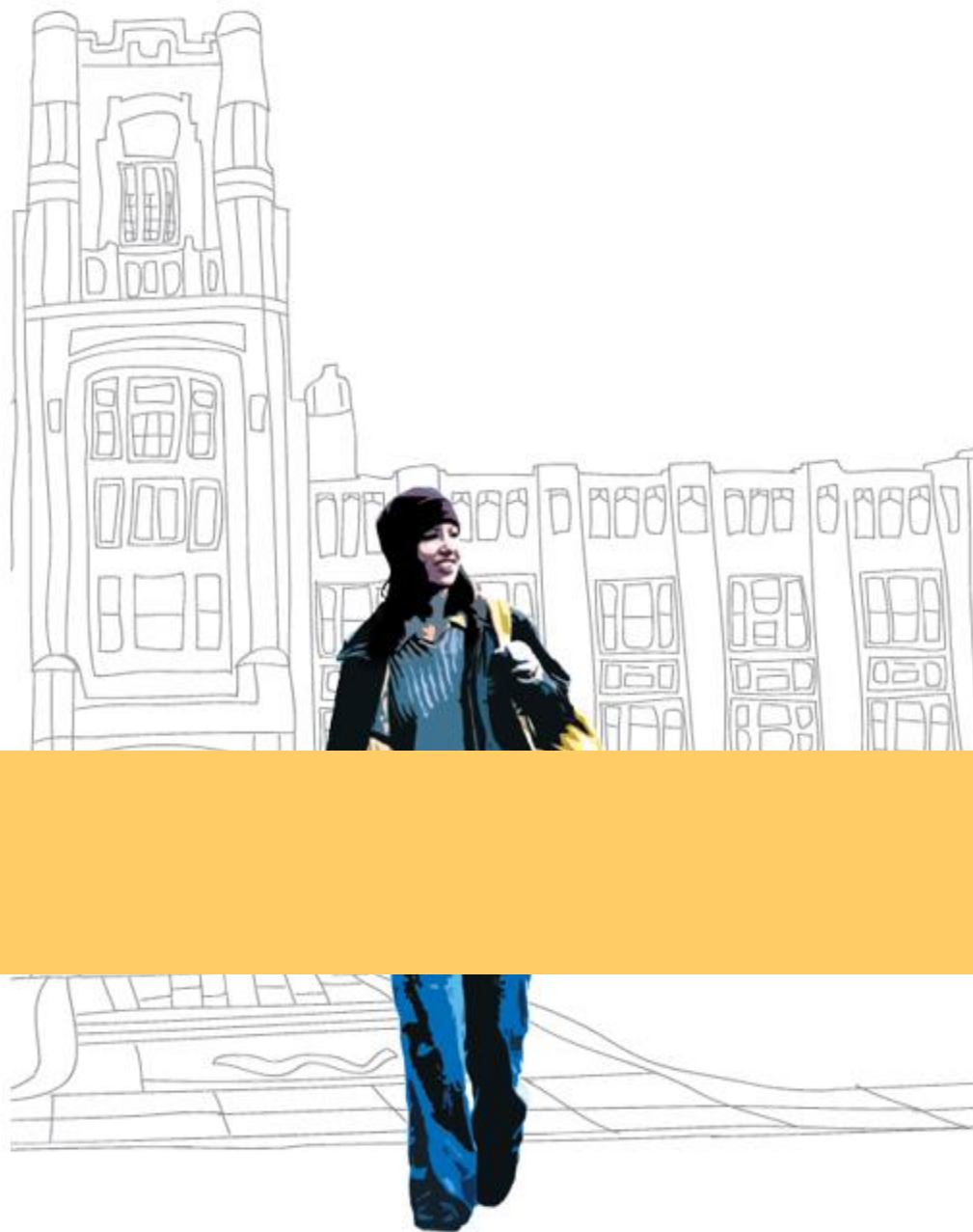


8

큐



## □ 이 장에서 다룰 내용

1

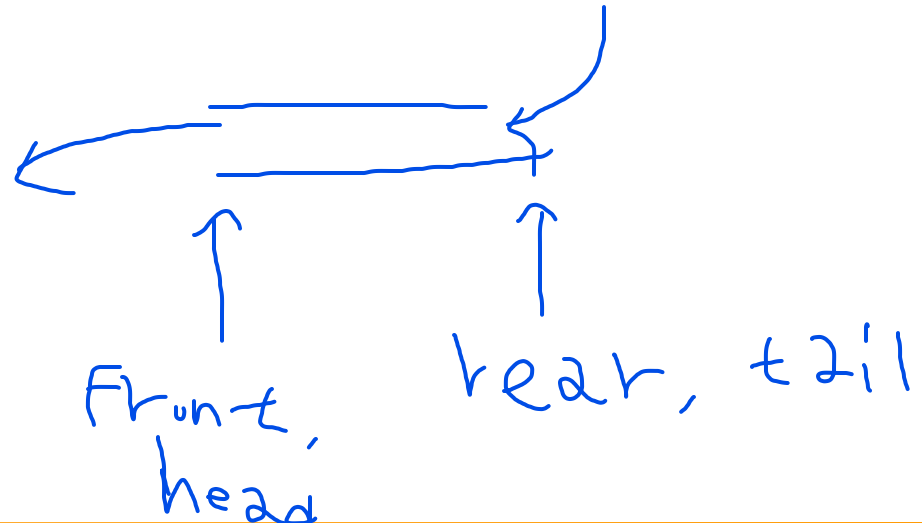
큐

2

큐의 구현

3

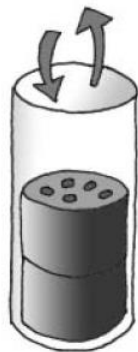
큐의 응용



## ❖ 큐(Queue)

- 스택과 마찬가지로 삽입과 삭제의 위치가 제한된 유한 순서 리스트
  - 큐의 뒤에서는 삽입만 하고, 앞에서는 삭제만 할 수 있는 구조
    - 삽입한 순서대로 원소가 나열되어 가장 먼저 삽입(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)된다.
- ☞ **선입선출 구조** (FIFO, First-In-First-Out)

### ■ 스택과 큐의 구조 비교



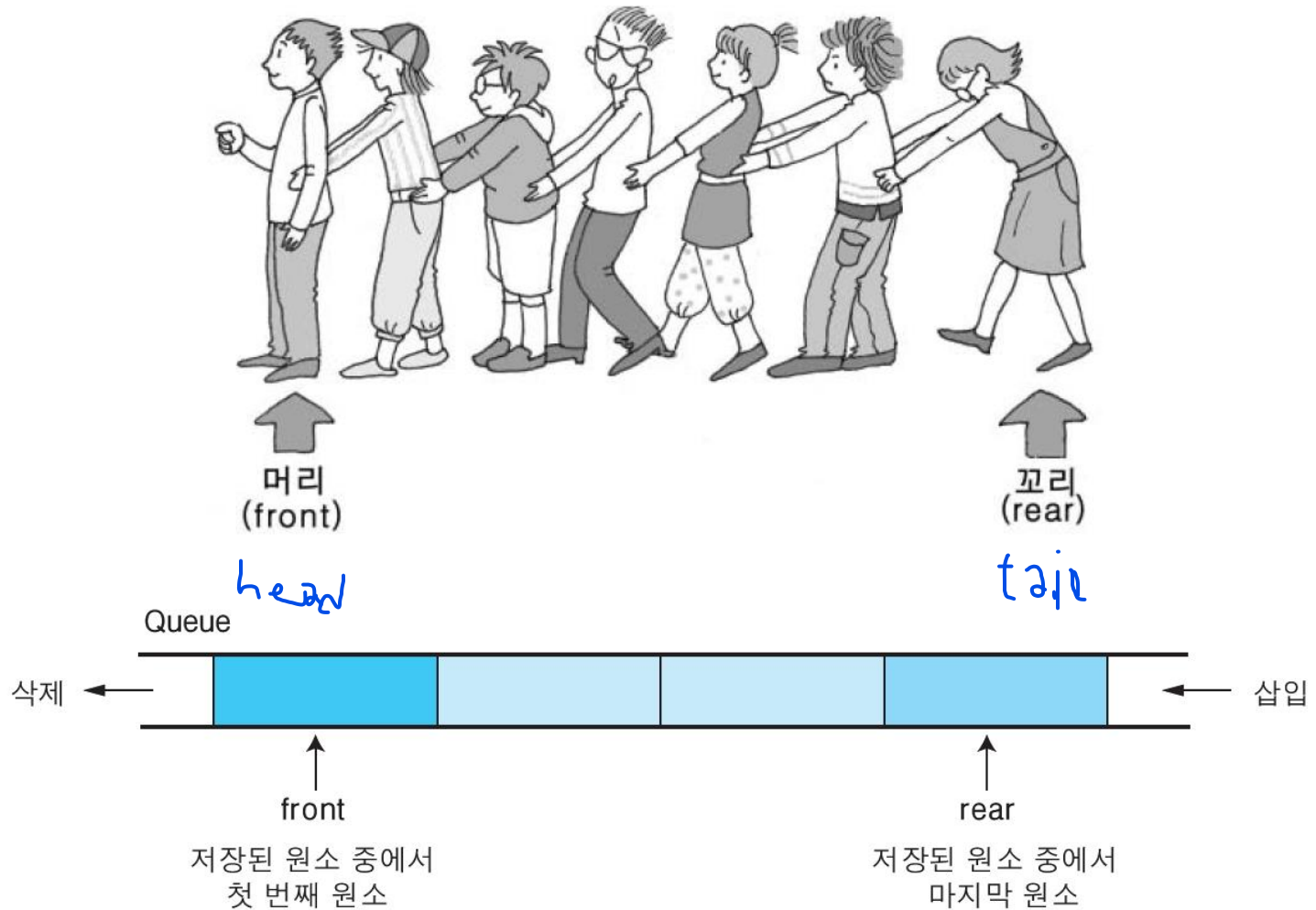
〈스택: 후입선출〉



〈큐: 선입선출〉



## ■ 큐의 구조





## ■ 큐의 연산

- 삽입 : enqueue
- 삭제 : dequeue

## ■ 스택과 큐의 연산 비교

자료구조 \ 항목	삽입 연산		삭제 연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	push	top	pop	top
큐	enqueue	rear	dequeue	front

## ■ 추상 자료형 큐

### ADT Queue

[ADT 8-1]

데이터 : 0개 이상의 원소를 가진 유한 순서 리스트

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

$\text{createQueue}() ::= \text{create an empty } Q$ ;

// 공백 큐를 생성하는 연산

$\text{isEmpty}(Q) ::= \text{if } (Q \text{ is empty}) \text{ then return true}$   
**else return false;**

// 큐가 공백인지 아닌지를 확인하는 연산

$\text{enqueue}(Q, \text{item}) ::= \text{insert item at the rear of } Q$ ;

// 큐의 rear에 item(원소)을 삽입하는 연산

$\text{dequeue}(Q) ::= \text{if } (\text{isEmpty}(Q)) \text{ then return error}$

**else { delete and return the front item of } Q };**

// 큐의 front에 있는 item(원소)을 큐에서 삭제하고 반환하는 연산

$\text{delete}(Q) ::= \text{if } (\text{isEmpty}(Q)) \text{ then return error}$

**else { delete the front item of } Q };**

// 큐의 front에 있는 item(원소)을 삭제하는 연산

$\text{peek}(Q) ::= \text{if } (\text{isEmpty}(Q)) \text{ then return error}$

**else { return the front item of the } Q };**

// 큐의 front에 있는 item(원소)을 반환하는 연산

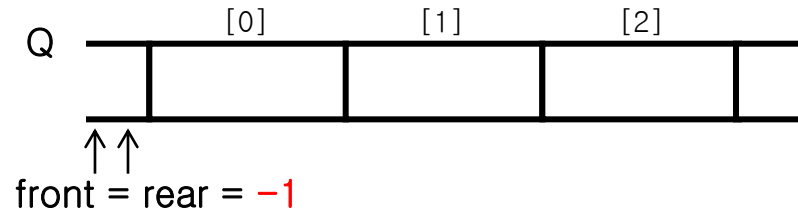
End Queue



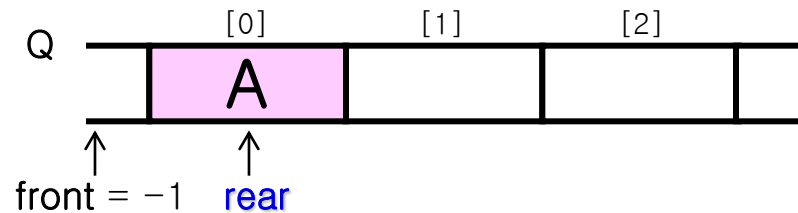
선형 Queue

## ■ 큐의 연산 과정

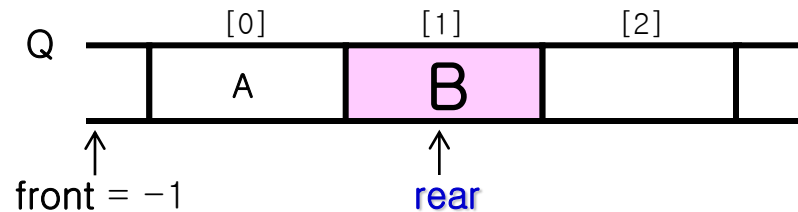
① 공백 큐 생성 : `createQueue();`



② 원소 A 삽입 : `enqueue(Q, A);`

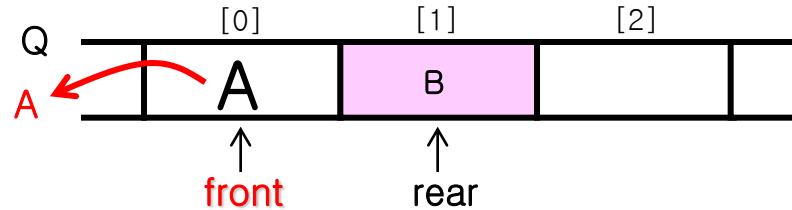


③ 원소 B 삽입 : `enqueue(Q, B);`

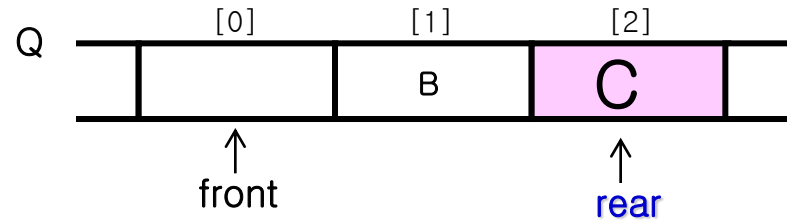




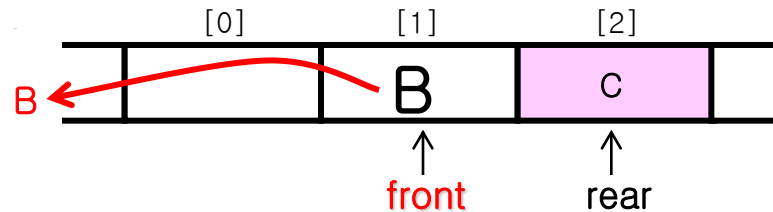
④ 원소 삭제 : `deQueue(Q);`



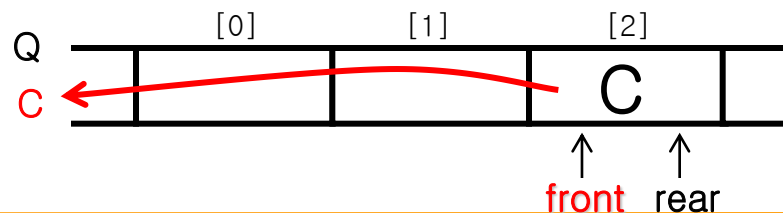
⑤ 원소 C 삽입 : `enqueue(Q, C);`



⑥ 원소 삭제 : `deQueue(Q);`



⑦ 원소 삭제 : `deQueue(Q);`





## □ 큐의 구현

### ❖ 선형 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)

## □ 큐의 구현

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

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

[알고리즘 8-1]

### ■ 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)

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

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

[알고리즘 8-2]

## ■ 큐의 삽입 알고리즘

```
enqueue(Q, item)
  if(isFull(Q)) then Queue_Full();
  else {
    rear ← rear+1; // ①
    Q[rear] ← item; // ②
  }
end enqueue()
```

[알고리즘 8-3]

- 마지막 원소의 뒤에 삽입해야 하므로

- ① 마지막 원소의 인덱스를 저장한 **rear**의 값을 하나 증가시켜 삽입할 자리 준비
- ② 그 인덱스에 해당하는 배열원소 **Q[rear]**에 **item**을 저장

## 큐의 삭제 알고리즘

```
deQueue(Q)
  if(isEmpty(Q)) then Queue_Empty();
  else {
    front ← front+1; // ①
    return Q[front]; // ②
  }
end deQueue()

delete(Q)
  if(isEmpty(Q)) then Queue_Empty();
  else front ← front+1;
end delete()
```

[알고리즘 8-4]

- 가장 앞에 있는 원소를 삭제해야 하므로

- ① front의 위치를 한자리 뒤로 이동하여 큐에 남아있는 첫 번째 원소의 위치로 이동하여 삭제할 자리 준비
- ② 그 자리의 원소를 삭제하여 반환

### ■ 큐의 검색 알고리즘

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

[알고리즘 8-5]

- 가장 앞에 있는 원소를 검색하여 반환하는 연산

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

## □ 큐의 구현

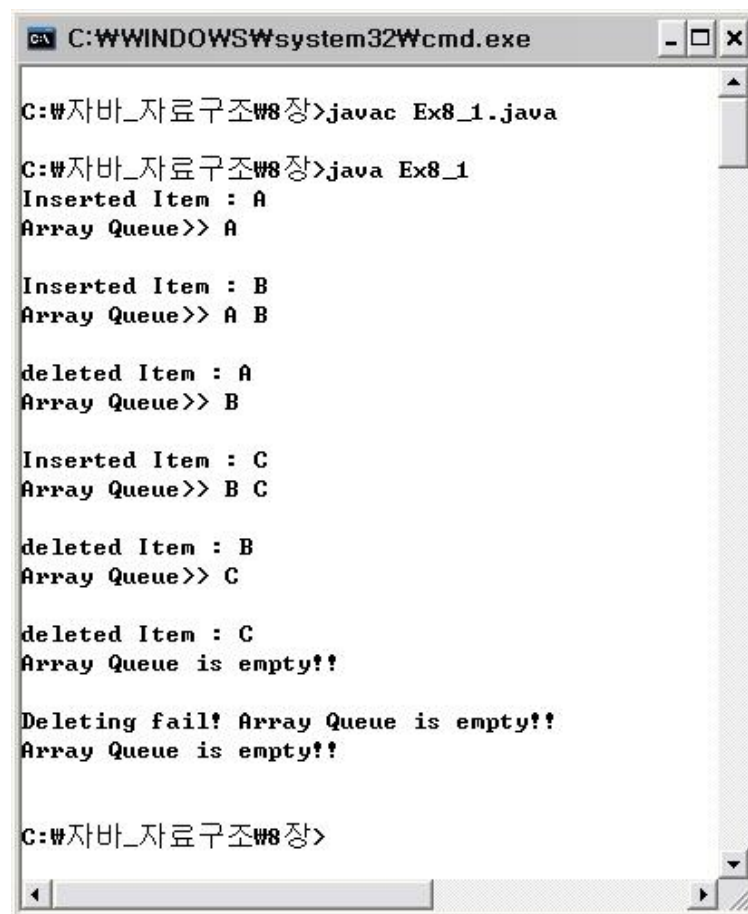
### ❖ 순차 자료구조 방식으로 구현한 큐 프로그램

[예제 8-1]

```
001 interface Queue{  
002     boolean isEmpty();  
003     void enqueue(char item);  
004     char dequeue();  
005     void delete();  
006     char peek();  
007 }  
008
```

>> 계속

## ■ 실행 결과



```
C:\WINDOWS\system32\cmd.exe

C:\자바_자료구조>javac Ex8_1.java

C:\자바_자료구조>java Ex8_1
Inserted Item : A
Array Queue>> A

Inserted Item : B
Array Queue>> A B

deleted Item : A
Array Queue>> B

Inserted Item : C
Array Queue>> B C

deleted Item : B
Array Queue>> C

deleted Item : C
Array Queue is empty!!

Deleting fail! Array Queue is empty!!
Array Queue is empty!!

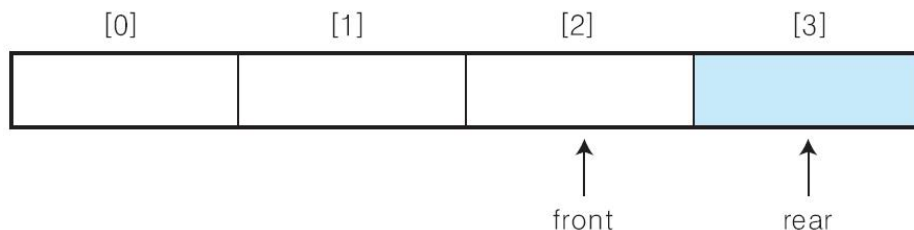
C:\자바_자료구조>
```



## ❖ 원형 큐

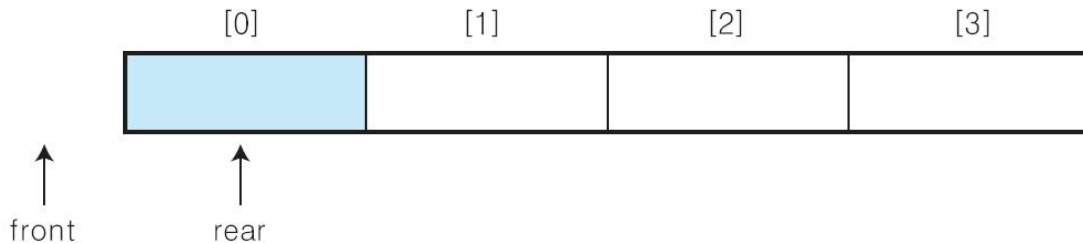
### ■ 선형 큐의 잘못된 포화상태 인식

- 큐에서 삽입과 삭제를 반복하면서 아래와 같은 상태일 경우, 앞부분에 빈 자리가 있지만  $rear=n-1$  상태이므로 포화상태로 인식하고 더 이상의 삽입을 수행하지 않는다.



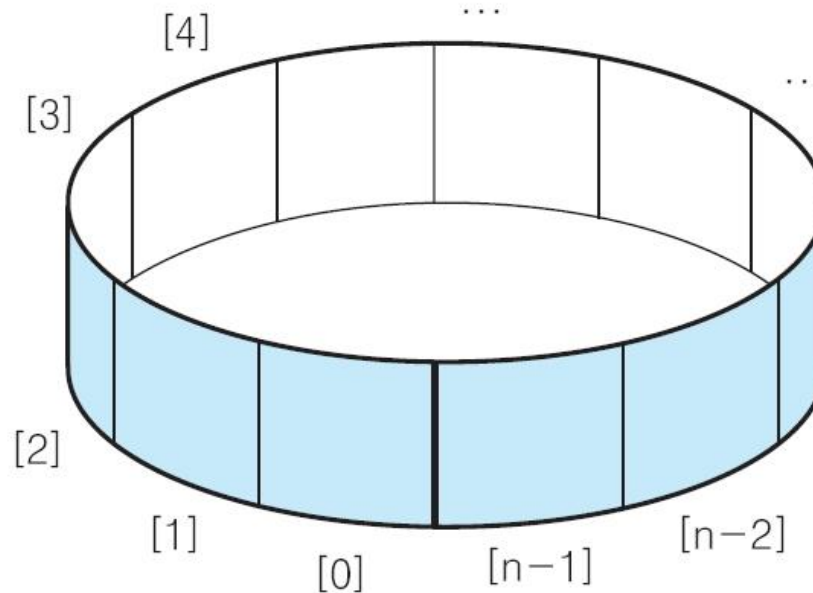
### ■ 선형 큐의 잘못된 포화상태 인식의 해결 방법-1

- 저장된 원소들을 배열의 앞부분으로 이동시키기
  - 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐



### ■ 선형 큐의 잘못된 포화상태 인식의 해결 방법-2

- 1차원 배열을 사용하면서 논리적으로 배열의 처음과 끝이 연결되어 있다고 가정하고 사용
- 원형 큐의 논리적 구조



### ■ 원형 큐의 구조

- 초기 공백 상태 :  $\text{front} = \text{rear} = 0$
- $\text{front}$ 와  $\text{rear}$ 의 위치가 배열의 마지막 인덱스  $n-1$ 에서 논리적인 다음자리인 인덱스 0번으로 이동하기 위해서 **나머지연산자 mod**를 사용
  - $3 \div 4 = 0 \dots 3$  (몫=0, 나머지=3)
  - $3 \bmod 4 = 3$
- 공백 상태와 포화 상태 구분을 쉽게 하기 위해서  $\text{front}$ 가 있는 자리는 사용하지 않고 항상 빈자리로 둔다.

### ■ 초기 공백 원형 큐 생성 알고리즘

- 크기가  $n$ 인 1차원 배열 생성
- front와 rear를 0으로 초기화

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

[알고리즘 8-6]

### ■ 원형 큐의 공백상태 검사 알고리즘과 포화상태 검사 알고리즘

- 공백 상태 :  $\text{front} = \text{rear}$
- 포화 상태 : 삽입할 rear의 다음 위치 = front의 현재 위치
  - $(\text{rear}+1) \bmod n = \text{front}$

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

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

[알고리즘 8-7]

### ■ 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  $\text{item}$ 을 삽입

```
enQueue(cQ, item)
  if(isFull(cQ)) then Queue_Full();
  else {
    rear  $\leftarrow$  (rear+1) mod n; // ①
    cQ[rear]  $\leftarrow$  item; // ②
  }
end enQueue()
```

[알고리즘 8-8]

## ■ 원형 큐의 삭제 알고리즘

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

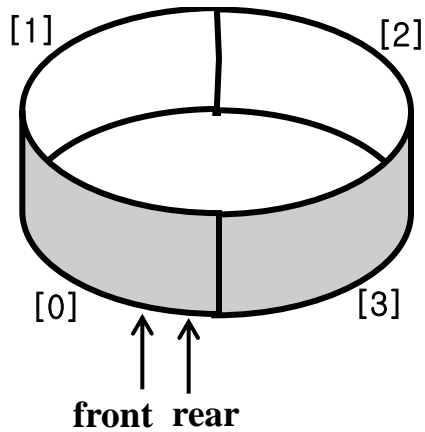
```
deQueue(cQ)
  if(isEmpty(cQ)) then Queue_Empty();
  else {
    front ← (front+1) mod n; // ①
    return cQ[front]; // ②
  }
end deQueue()

delete(cQ)
  if(isEmpty(Q)) then Queue_Empty();
  else front ← (front+1) mod n;
end delete()
```

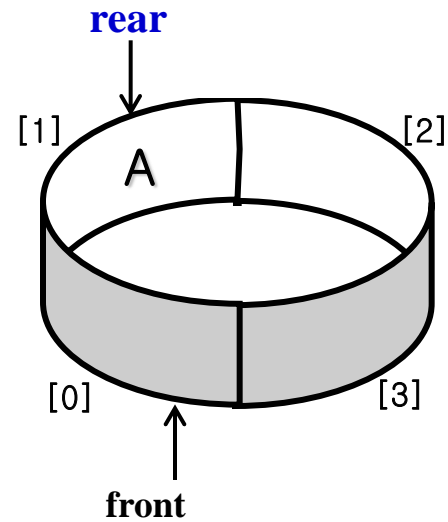
[알고리즘 8-9]

## ■ 원형 큐에서의 연산 과정

① createQueue();



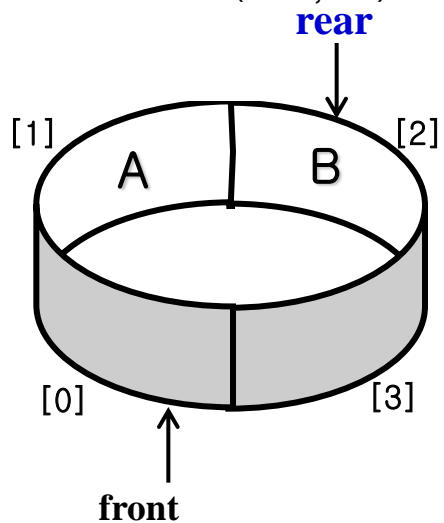
② enqueue(cQ, A);



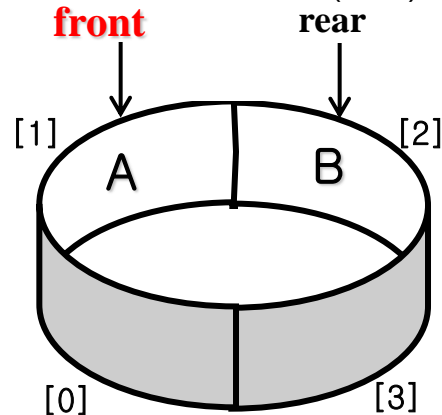


## 큐의 구현

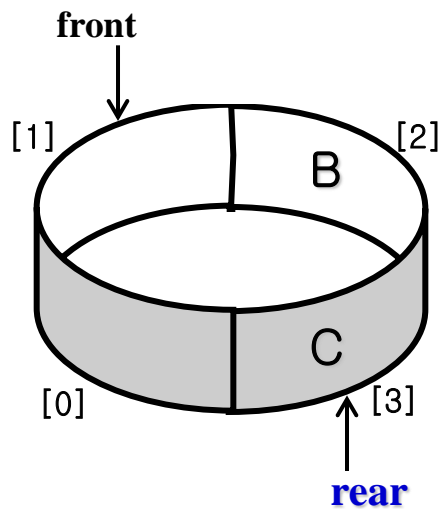
③ enQueue(cQ, B);



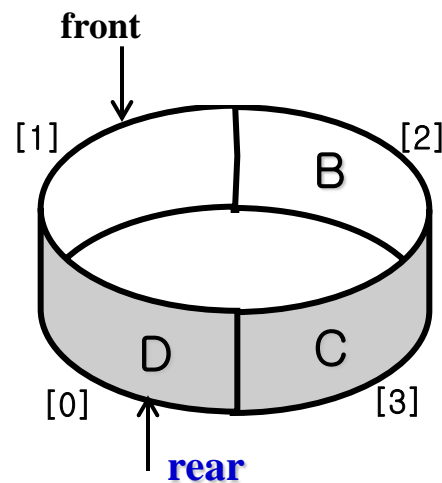
④ deQueue(cQ);



⑤ enQueue(cQ, C);



⑥ enQueue(cQ, D);



## □ 큐의 구현

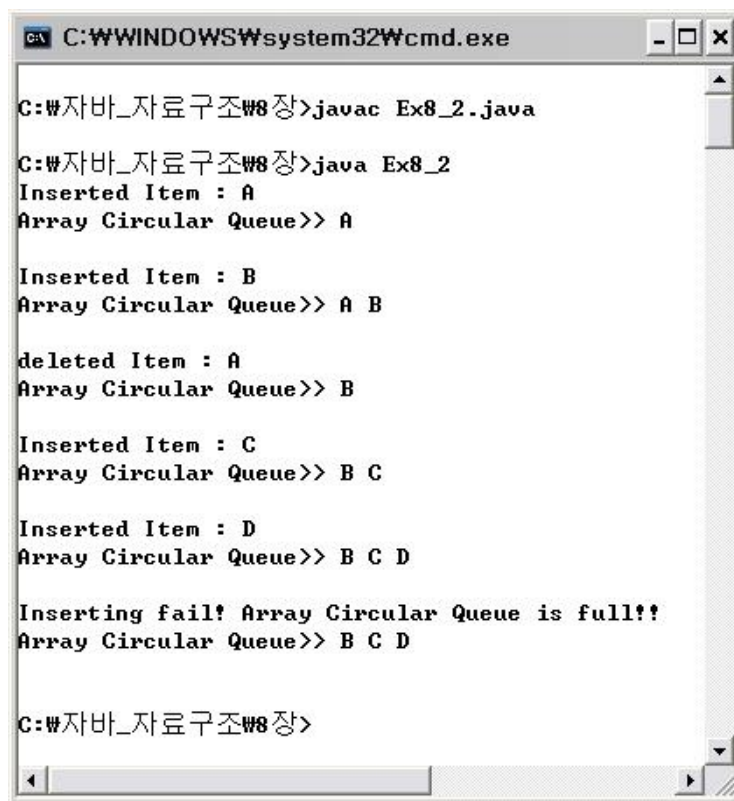
### ❖ 순차 자료구조 방식으로 구현한 원형 큐 프로그램

[예제 8-2]

```
001 interface Queue{  
002     boolean isEmpty();  
003     void enqueue(char item);  
004     char dequeue();  
005     void delete();  
006     char peek();  
007 }  
008
```

>> 계속

## ■ 실행 결과



```
C:\WINDOWS\system32\cmd.exe

C:\자바_자료구조\8>javac Ex8_2.java

C:\자바_자료구조\8>java Ex8_2
Inserted Item : A
Array Circular Queue>> A

Inserted Item : B
Array Circular Queue>> A B

deleted Item : A
Array Circular Queue>> B

Inserted Item : C
Array Circular Queue>> B C

Inserted Item : D
Array Circular Queue>> B C D

Inserting fail! Array Circular Queue is full!!
Array Circular Queue>> B C D

C:\자바_자료구조\8>
```

# 큐의 구현

## ❖ 연결 큐

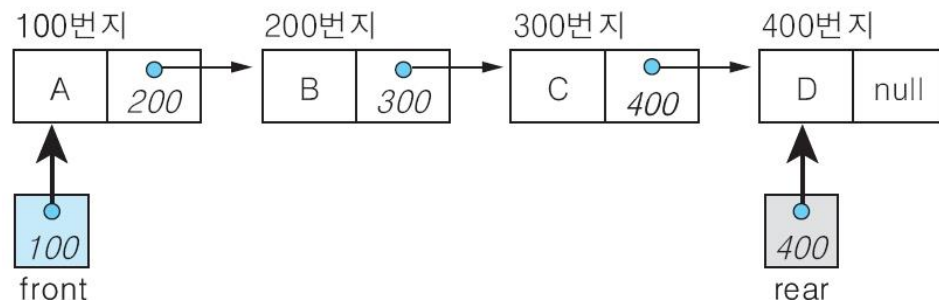
### ■ 단순 연결 리스트를 이용한 큐

- 큐의 원소 : 단순 연결 리스트의 노드
- 큐의 원소의 순서 : 노드의 링크 포인터로 연결
- 변수 front : 첫 번째 노드를 가리키는 포인터 변수
- 변수 rear : 마지막 노드를 가리키는 포인터 변수

### ■ 상태 표현

- 초기 상태와 공백 상태 : front = rear = null

### ■ 연결 큐의 구조



### ■ 초기 공백 연결 큐 생성 알고리즘

- 초기화 : front = rear = null

```
createLinkedQueue()  
    front ← null;  
    rear ← null;  
end createLinkedQueue()
```

[알고리즘 8-10]

### ■ 공백 연결 큐 검사 알고리즘

- 공백 상태 : front = rear = null

```
isEmpty(LQ)
  if(front=null) then return true;
  else return false;
end isEmpty()
```

[알고리즘 8-11]

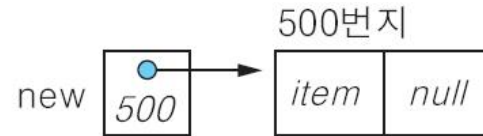
### ■ 연결 큐의 삽입 알고리즘

```
enqueue(LQ, item)
  new ← getNode();
  new.data ← item; // ❶
  new.link ← null;
  if (isEmpty(LQ)) then { // ❷ 연결 큐가 공백인 경우
    rear ← new;
    front ← new;
  }
  else { // ❸ 연결 큐에 노드가 있는 경우
    rear.link ← new;
    rear ← new;
  }
end enqueue()
```

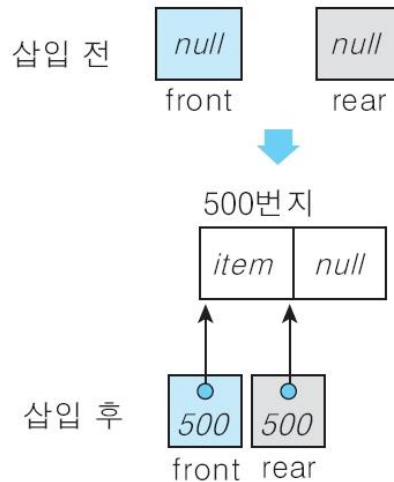
[알고리즘 8-12]

## □ 큐의 구현

- ① 삽입할 새 노드를 생성하여 데이터 필드에 `item`을 저장한다. 삽입할 새 노드는 연결 큐의 마지막 노드가 되어야 하므로 링크 필드에 `null`을 저장한다.



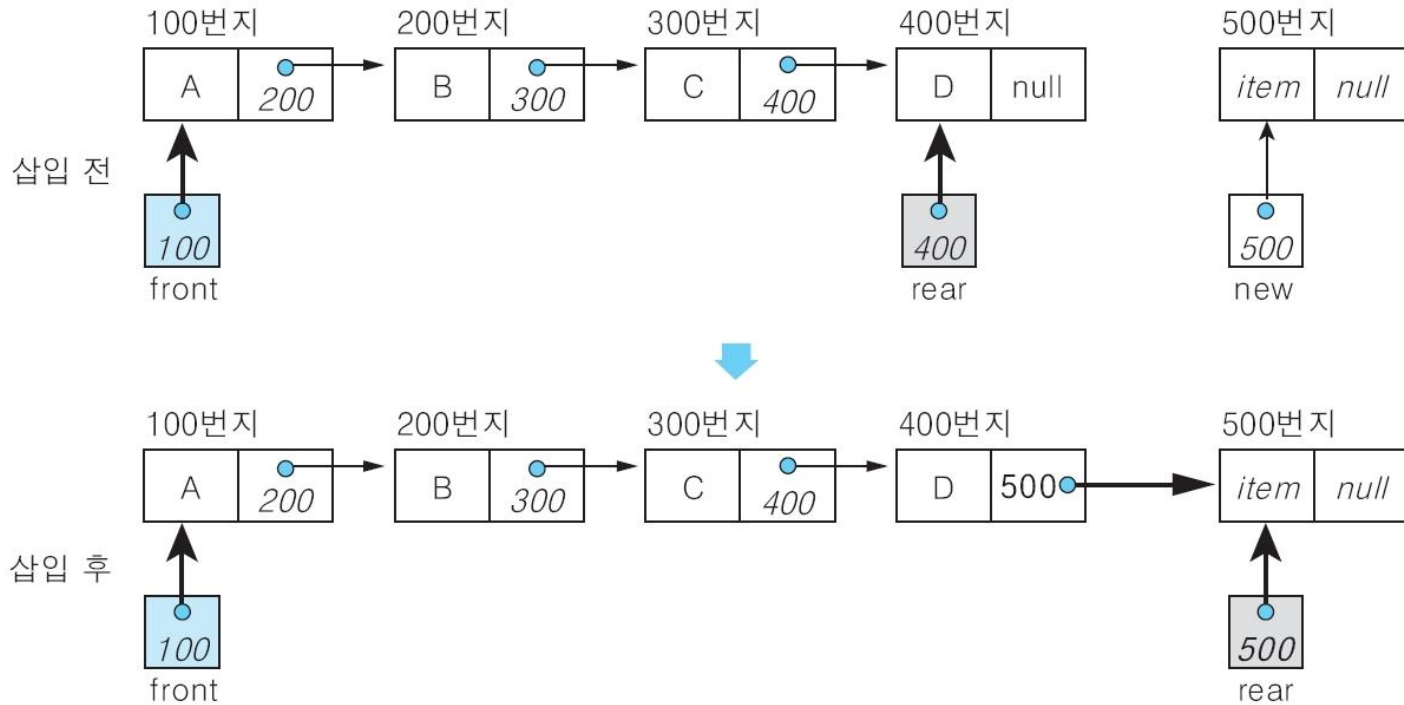
- ② 새 노드를 삽입하기 전에 연결 큐가 공백인지 아닌지를 검사한다. 연결 큐가 공백인 경우에는 삽입할 새 노드가 큐의 첫 번째 노드이자 마지막 노드이므로 포인터 `front`와 `rear`가 모두 새 노드를 가리키도록 설정한다.





## 큐의 구현

- ③ 큐가 공백이 아닌 경우, 즉 노드가 있는 경우에는 현재 큐의 마지막 노드의 뒤에 새 노드를 삽입하고 마지막 노드를 가리키는 rear가 삽입한 새 노드를 가리키도록 설정한다.



## ■ 연결 큐의 삭제 알고리즘

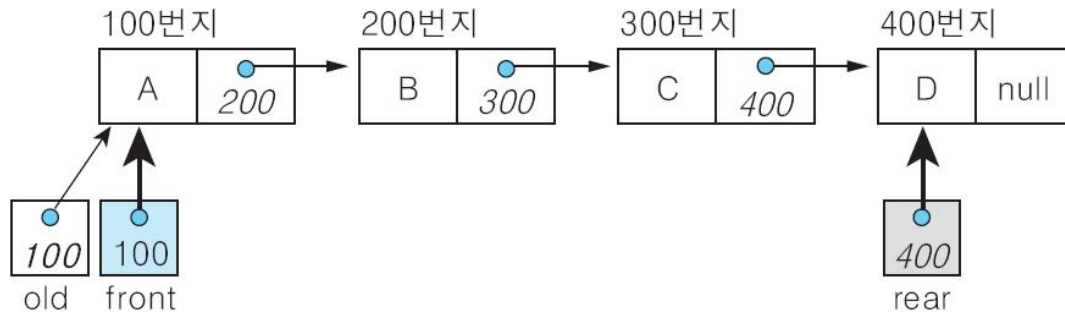
```
deQueue(LQ)
  if(isEmpty(LQ)) then Queue_Empty();
  else {
    old ← front; // ❶
    item ← front.data;
    front ← front.link; // ❷
    if (isEmpty(LQ)) then rear ← null; // ❸
    returnNode(old); // ❹
    return item;
  }
end deQueue()

delete(LQ)
  if(isEmpty(LQ)) then Queue_Empty();
  else {
    old ← front;
    front ← front.link;
    if(isEmpty(LQ)) then rear ← null;
    returnNode(old);
  }
end delete()
```

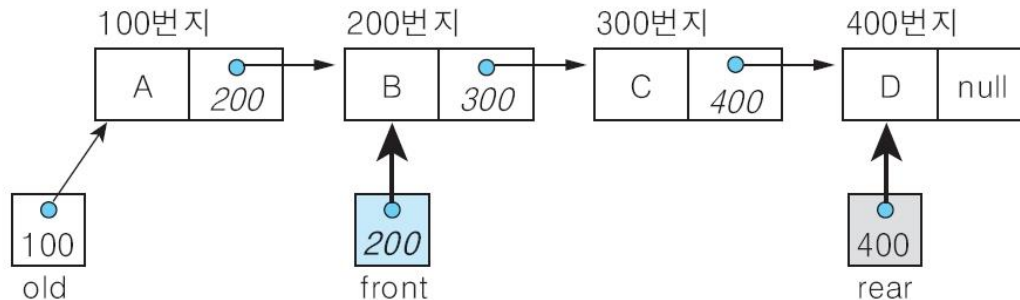
[알고리즘 8-13]

## □ 큐의 구현

- ① 삭제연산에서 삭제할 노드는 큐의 첫 번째 노드로서 포인터 front가 가리키고 있는 노드이다. front가 가리키는 노드를 포인터 old가 가리키게 하여 삭제할 노드를 지정한다.



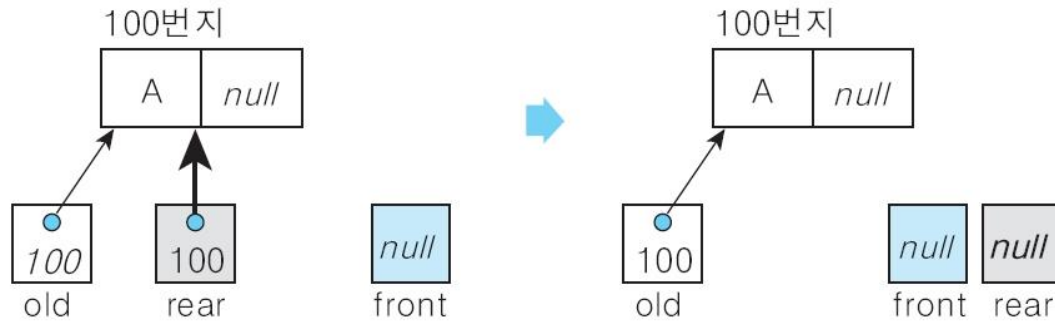
- ② 삭제연산 후에는 현재 front 노드의 다음 노드가 front 노드(첫번째 노드)가 되어야하므로, 포인터 front를 재설정한다.



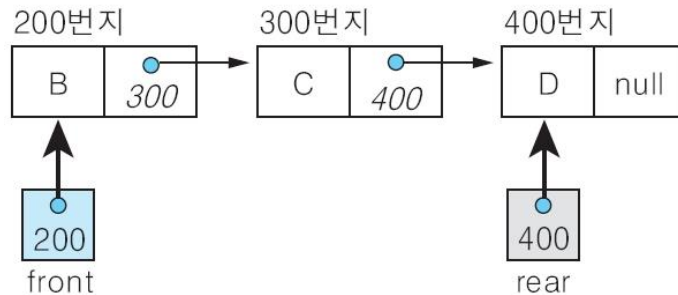
## □ 큐의 구현

③ 현재 큐에 노드가 하나뿐이어서 삭제연산 후에 공백 큐가 되는 경우 :

☞ 큐의 마지막 노드가 없어지므로 포인터 rear를 null로 설정한다.



④ 포인터 old가 가리키고 있는 노드를 삭제하고, 메모리 공간을 시스템에 반환(returnNode())한다



### ■ 연결 큐의 검색 알고리즘

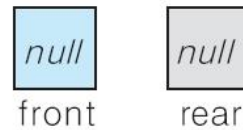
- 연결 큐의 첫 번째 노드, 즉 front 노드의 데이터 필드 값을 반환

```
peek(LQ)
  if(isEmpty(LQ)) then Queue_Empty()
  else return (front.data);
end peek()
```

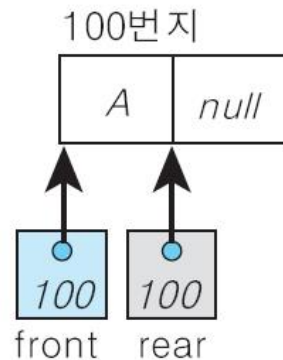
[알고리즘 8-14]

## ■ 연결 큐에서의 연산 과정

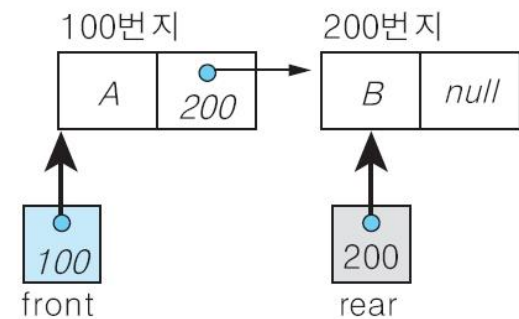
① 공백 큐 생성 : `createLinkedListQueue();`



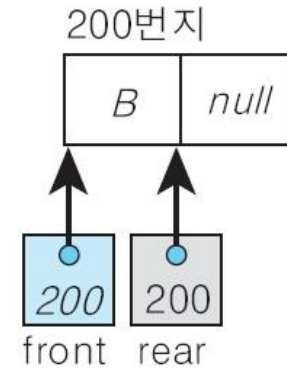
② 원소 A 삽입 : `enqueue(LQ, A);`



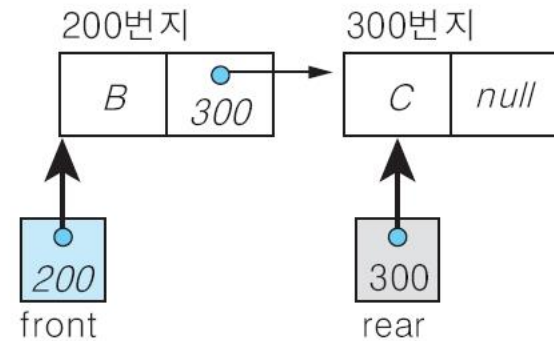
③ 원소 B 삽입 : `enqueue(LQ, B);`



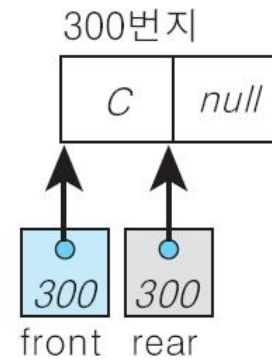
④ 원소 삭제 : `deQueue(LQ);`



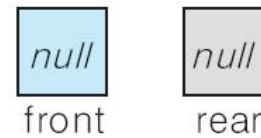
⑤ 원소 C 삽입 : `enqueue(LQ, C);`



⑥ 원소 삭제 : `deQueue(LQ);`



⑦ 원소 삭제 : `deQueue(LQ);`





- 연결 자료구조 방식을 이용하여 구현한 연결 큐 프로그램

[예제 8-3]

```
001 interface Queue{
002     boolean isEmpty();
003     void enqueue(char item);
004     char dequeue();
005     void delete();
006     char peek();
007 }
008
```

# □ 큐의 응용

## ❖ 운영체제의 작업 큐

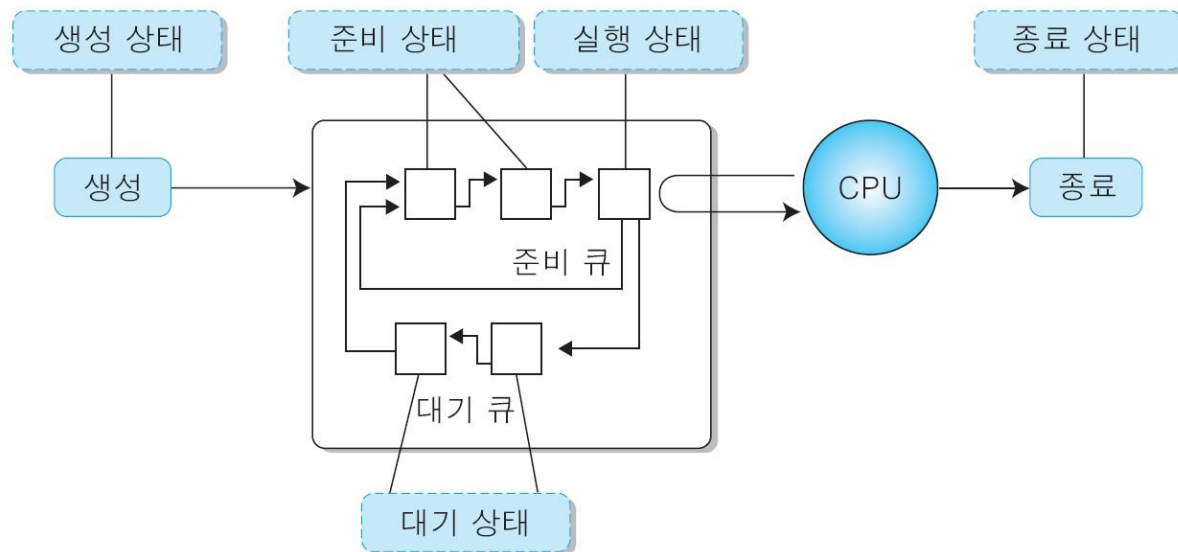
### ■ 프린터 버퍼 큐

- CPU에서 프린터로 보낸 데이터 순서대로(선입선출) 프린터에서 출력하기 위해서 선입선출 구조의 큐 사용

### ■ 스케줄링 큐

- CPU 사용을 요청한 프로세서들의 순서를 스케줄링하기 위해 큐를 사용

프린터  
버퍼  
프린터



## □ 큐의 응용

### ❖ 시뮬레이션 큐잉 시스템

- 시뮬레이션을 위한 수학적 모델링에서 대기행렬과 대기시간 등을 모델링하기 위해서 큐잉 이론(Queue theory) 사용

A woman with dark hair, wearing a dark jacket over a blue and white striped shirt, is walking from left to right. She is carrying a yellow bag. The background is a line drawing of a building with many windows. The text "Thank You !" is overlaid on a yellow banner across the middle of the image.

**Thank You !**