1 큐의 개념과 특징

1 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(First-In)한 원소는 맨 앞에 있다가 가장 먼저 삭제(First-Out)됨
    → 선입선출 구조(FIFO, First-In-First-Out)

1 큐(Queue)

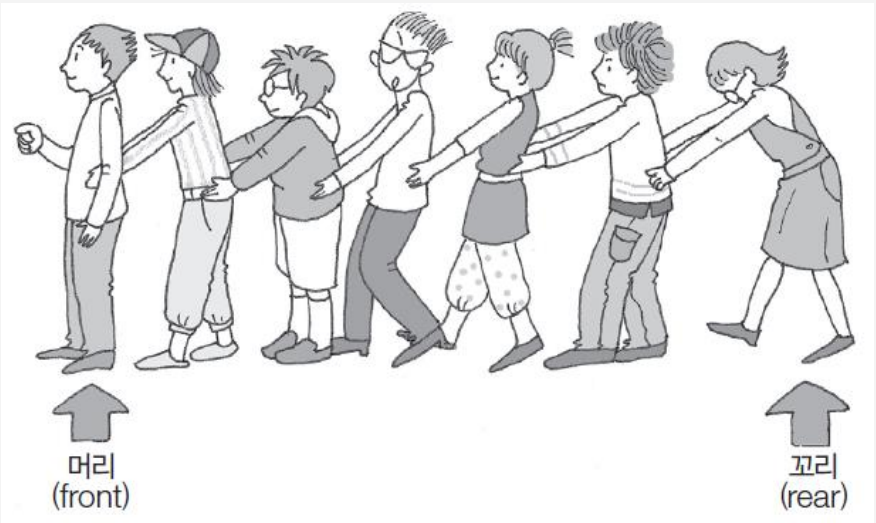
[스택과 큐의 구조 비교 예]

(a) 스택의 구조 (b) 큐의 구조

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

2 FIFO 구조의 예

[꼬리잡기 놀이의 머리와 꼬리]



※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

2 FIFO 구조의 예

[큐의 FIFO 구조]

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

3 큐의 연산

- 삽입 : enQueue
- 삭제 : deQueue

4 스택과 큐의 연산 비교

항목 / 자료 구조	삽입 연산		삭제연산	
	연산자	삽입 위치	연산자	삭제 위치
스택	push	top	pop	top
큐	enQueue	rear	deQueue	front

5 큐의 추상 자료형

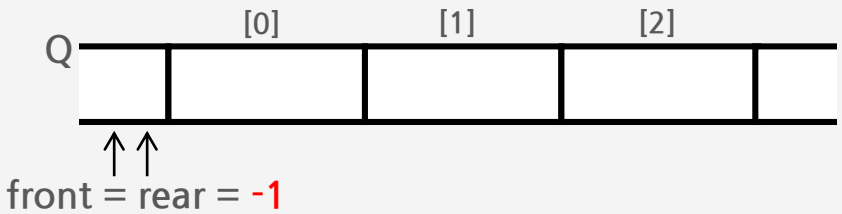
ADT 6-1 큐의 추상 자료형

```
ADT Queue
데이터 : 0개 이상의 원소를 가진 유한 순서 리스트
연산 : Q ∈ Queue; item ∈ 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
```

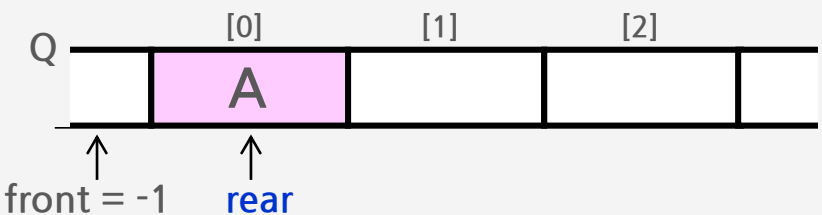
※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

6 큐의 연산 과정

① 공백 큐 생성 : createQueue();

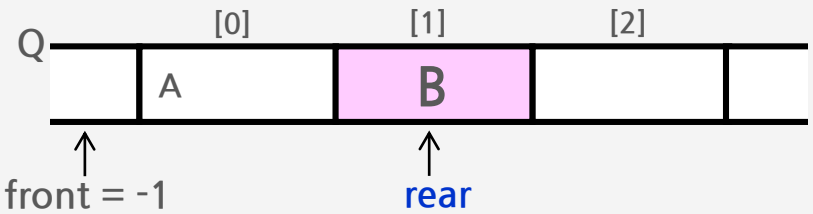


② 원소 A 삽입 : enQueue(Q, A);

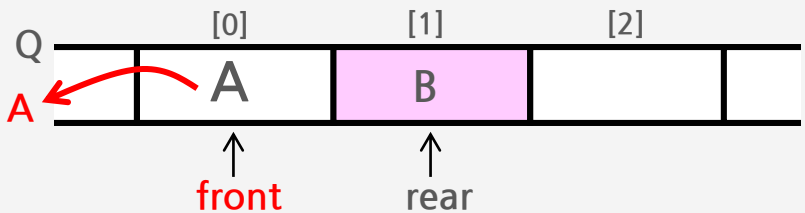


6 큐의 연산 과정

③ 원소 B 삽입 : enQueue(Q, B);

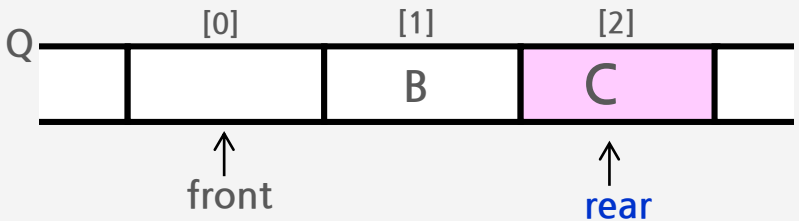


④ 원소 삭제 : deQueue(Q)

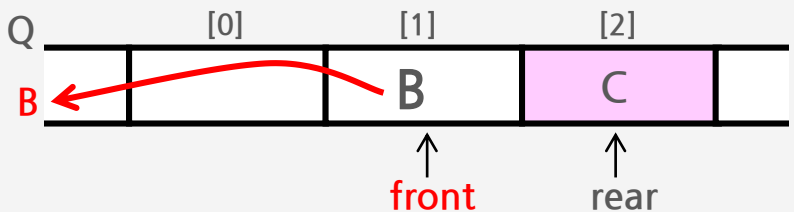


6 큐의 연산 과정

⑤ 원소 C 삽입 : enQueue(Q, C)

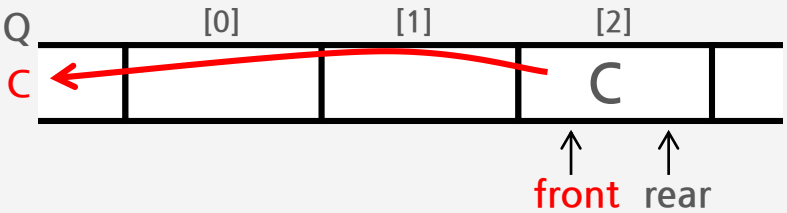


⑥ 원소 삭제 : deQueue(Q);



6 큐의 연산 과정

⑦ 원소 삭제 : deQueue(Q);



2 순차 큐의 구현

1 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

2 | 순차 큐의 구현

1 순차 큐

- 상태 표현
  - 초기 상태 : front = rear = -1
  - 공백 상태 : front = rear
  - 포화 상태 : rear = n-1
    (n : 배열의 크기, n-1 : 배열의 마지막 인덱스)

2 C언어 구조체로 순차 큐 자료구조 정의

```
typedef char element;       // 큐 원소(element)의 자료형을 char로 정의
typedef struct {
            element queue[Q_SIZE];   // 1차원 배열 큐 선언
            int front, rear;
} QueueType;
```

3 초기 공백 큐 생성 알고리즘

- 크기가 n인 1차원 배열 생성
- front와 rear를 -1로 초기화

알고리즘 6-1 공백 순차 큐 생성

```
createQueue()
    Q[n];
    front ← -1;
    rear ← -1;
end createQueue()
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

3 초기 공백 큐 생성 알고리즘

- C언어로 공백 순차 큐를 생성하는 연산

```
QueueType *createQueue() {
        QueueType *Q;
        Q = (QueueType *)malloc(sizeof(QueueType));
        Q->front = -1;  // front 초깃값 설정
        Q->rear = -1;    // rear 초깃값 설정
        return Q;
}
```

4 공백 큐 검사 알고리즘과 포화상태 검사 알고리즘

- 공백 상태 : front = rear
- 포화 상태 : rear = n-1
  (n : 배열의 크기, n-1 : 배열의 마지막 인덱스)

4 공백 큐 검사 알고리즘과 포화상태 검사 알고리즘

알고리즘 6-2 순차 큐 공백 상태 검사

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

알고리즘 6-3 순차 큐의 포화 상태 검사

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

```
// C언어로 순차 큐가 공백 상태인지 검사하는 연산
int isEmpty(QueueType *Q) {
        if (Q->front == Q->rear) {
                printf(" Queue is empty! ");
                return 1;
        }
        else return 0;
}

// C언어로 순차 큐가 포화 상태인지 검사하는 연산
int isFull(QueueType *Q) {
        if (Q->rear == Q_SIZE - 1) {
                printf(" Queue is full! ");
                return 1;
        }
        else return 0;
}
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

5 큐의 삽입 알고리즘

알고리즘 6-4 순차 큐의 원소 삽입

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

```
// C언어로 순차 큐의 rear에 원소를 삽입하는 연산
void enQueue(QueueType *Q, element item) {
    if (isFull(Q)) return;  // 포화 상태이면, 삽입 연산 중단
    else {
            Q->rear++;
            Q->queue[Q->rear] = item;
     }
}
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

- 마지막 원소의 뒤에 삽입해야 하므로
  ① 마지막 원소의 인덱스를 저장한 rear의 값을 하나 증가시켜 삽입할 자리 준비
  ② 수정한 rear값에 해당하는 배열원소 Q[rear]에 item을 저장

6 큐의 삭제 알고리즘

알고리즘 6-5 순차 큐의 원소 삭제

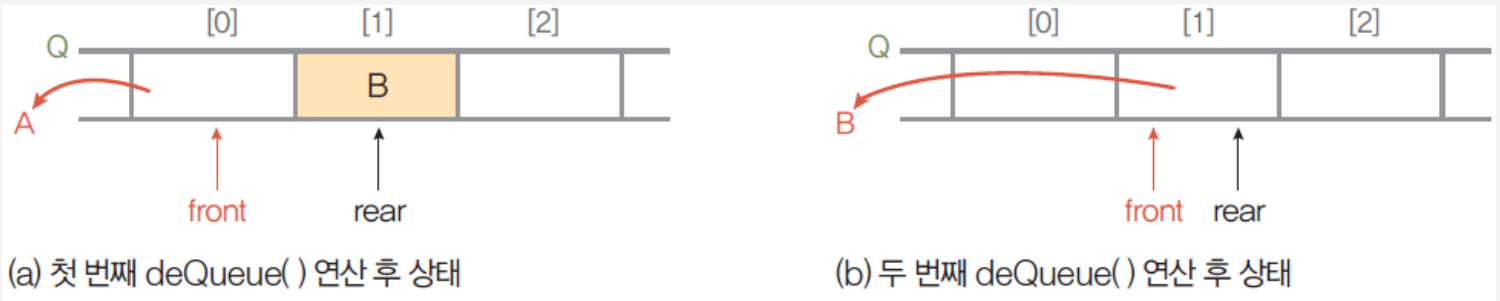
```
deQueue(Q)
    if (isEmpty(Q)) then Queue_Empty();
    else {     // 공백 상태이면 삭제 연산 중단
        ❶ front ← front + 1;
        ❷ return Q[front];
    }
end deQueue()
```

```
// C언어에서 순차 큐의 front에서 원소를 삭제하는 연산
element deQueue(QueueType *Q) {
    if (isEmpty(Q)) return;  // 공백 상태이면, 삭제 연산 중단
    else {
        Q->front++;
        return Q->queue[Q->front];
    }
}
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

- 가장 앞에 있는 원소를 삭제해야 하므로
  1. front의 위치를 한자리 뒤로 이동하여 큐에 남아있는 첫 번째 원소의 위치로 이동하여 삭제할 자리 준비
  2. front 자리의 원소를 삭제하여 반환

7 큐의 삭제 후 상태



[deQueue( ) 연산 후 상태]

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

8 Self Test

- 공백 순차 큐(크기 5인 1차원 배열)에서 다음 연산의 수행 결과 상태를 설명하세요.

  풀이)
  enQueue(Q1, A), enQueue(Q1, B),
  enQueue(Q1, C), deQueue(Q1),enQueue(Q1, D),
  deQueue(Q1), deQueue(Q1), enQueue(Q1, E)

9 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()
```

```
// C언어에서 순차 큐의 가장 앞에 있는 원소를 검색하는 연산
element peek(QueueType *Q) {
    if (isEmpty(Q)) exit(1); // 공백 상태이면 연산 중단
    else return Q->queue[Q->front + 1];
}
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

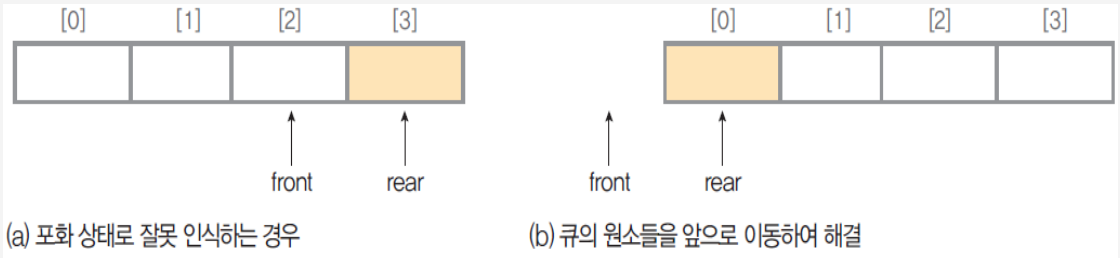
- 가장 앞에 있는 원소를 검색하여 반환하는 연산
  - ① 현재 front의 한자리 뒤(front+1)에 있는 원소, 즉 큐에 있는 첫 번째 원소를 반환

3 원형 큐의 구현

3 원형 큐의 구현

1 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 않음



[순차 큐의 잘못된 포화 상태 문제와 해결 방법]

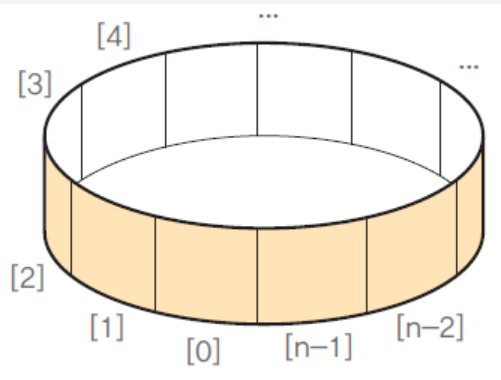
※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

2 순차 큐의 잘못된 포화상태 인식의 해결 방법 (1)

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

3 순차 큐의 잘못된 포화상태 인식의 해결 방법 (2)

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



※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

4 원형 큐의 구조

- 초기 공백 상태 : front = rear = 0
- front와 rear의 위치가 배열의 마지막 인덱스 n-1에서 논리적인 다음 자리인 인덱스 0번으로 이동하기 위해서 **나머지 연산자 mod**를 사용
  - 3 ÷ 4 = 0 …**3** (몫=0, **나머지=3**)
  - 3 mod 4 = **3**

4 원형 큐의 구조

- 순차 큐와 원형 큐의 비교

종류	삽입 위치	삭제 위치
순차 큐	rear = rear + 1	front = front +1
큐	rear = (rear +1)mod n	front = (front+1)mod n

- 사용조건) 공백 상태와 포화 상태 구분을 쉽게 하기 위해서 front가 있는 자리는 사용하지 않고 항상 빈자리로 둠

5 초기 공백 원형 큐 생성 알고리즘

- 크기가 n인 1차원 배열 생성
- front와 rear를 0 으로 초기화

알고리즘 6-7 공백 원형 큐 생성

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

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

5 초기 공백 원형 큐 생성 알고리즘

- C언어로 공백 원형 큐를 생성하는 연산

```
QueueType *createQueue() {
        QueueType *cQ;
        cQ = (QueueType *)malloc(sizeof(QueueType));
        cQ->front = 0;      // front 초깃값 설정
        cQ->rear = 0;       // rear 초깃값 설정
        return cQ;
}
```

6 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()
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

6 원형 큐의 공백상태 검사 알고리즘과 포화상태 검사 알고리즘

- 원형 큐의 상태에 따른 front와 rear의 관계

구분	조건
공백 상태	front = rear
포화 상태	(rear+1)mod n = front

7 원형 큐의 삽입 알고리즘

- rear의 값을 조정하여 삽입할 자리를 준비
  : rear ← (rear+1) mod n;

- 준비한 자리 cQ[rear]에 원소 item을 삽입

알고리즘 6-10 원형 큐의 원소 삽입

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

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

8 원형 큐의 삭제 알고리즘

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

알고리즘 6-11 원형 큐의 원소 삭제

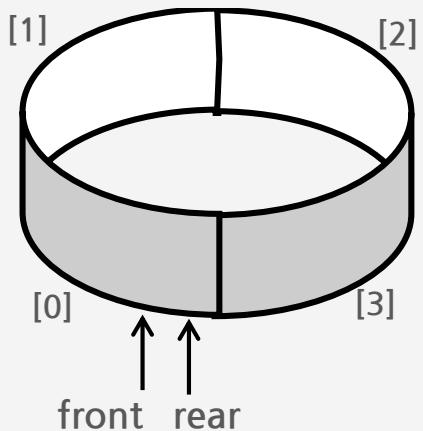
```
deQueue(cQ)
    if (isEmpty(cQ)) then Queue_Empty();
    else {  // 공백 상태이면 삭제 연산 중단
        ❶ front ← (front + 1) mod n;
        ❷ return cQ[front];
}
end deQueue()
```

※출처: 이지영(2016). IT CookBook, C로 배우는 쉬운 자료구조(개정3판). 한빛미디어

9 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정

① 공백 원형 큐 생성 : createQueue();

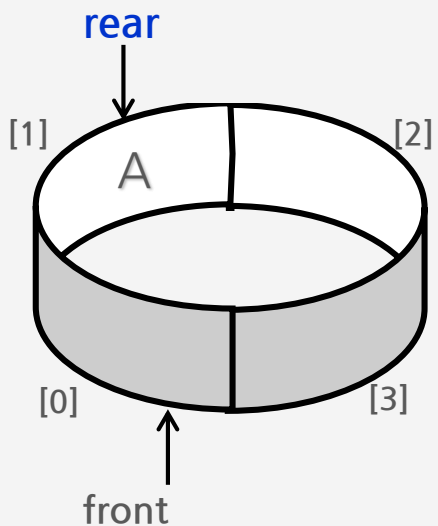
```
QueueType *createQueue() {
    QueueType *cQ;
    cQ = (QueueType *)malloc(sizeof(QueueType));
    cQ->front = 0;      // front 초깃값 설정
    cQ->rear = 0;       // rear 초깃값 설정
    return cQ;
 }
```



9 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정

② 원소 A 삽입 : enQueue(cQ, A);

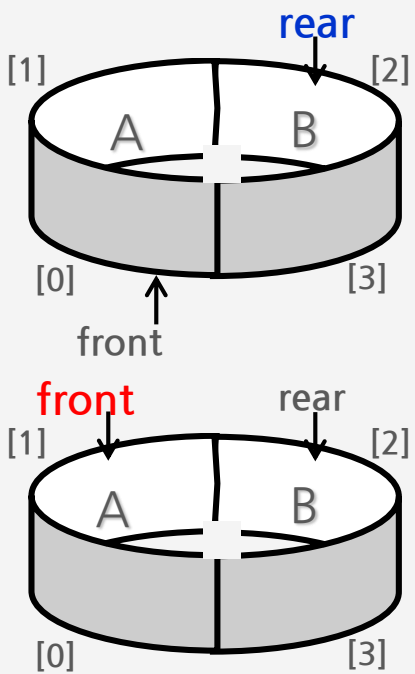
```
// 원형 큐가 포화 상태인지 검사하는 연산
int isFull(QueueType *cQ) {
        if (((cQ->rear + 1) % cQ_SIZE) == cQ->front) {
           printf("  Circular Queue is full! ");
           return 1;
        }else return 0;
}
void enQueue(QueueType *cQ, element item) {
        if (isFull(cQ))  return;
        else {
            cQ->rear = (cQ->rear + 1) % cQ_SIZE;
            cQ->queue[cQ->rear] = item;
        }
}
```



9 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정

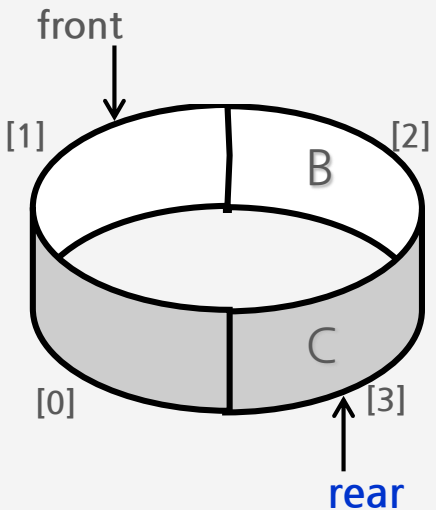
③ 원소 B 삽입 : enQueue(cQ, B);
④ 원소 삭제 : deQueue(cQ);(삭제 데이터 : A)

```
// 원형 큐가 공백 상태인지 검사하는 연산
int isEmpty(QueueType *cQ) {
        if (cQ->front == cQ->rear) {
          printf(" Circular Queue is empty! ");
          return 1;
        }
        else return 0;
}

element deQueue(QueueType *cQ) {
        if (isEmpty(cQ)) exit(1);
        else {
          cQ->front = (cQ->front + 1) % cQ_SIZE;
          return cQ->queue[cQ->front];
        }
}
```



9 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정

⑤ 원소 C 삽입 : enQueue(cQ, C);



9 크기가 4인 원형 큐에서 큐를 생성하고 삽입·삭제하는 연산 과정

⑥ 원소 D 삽입 : enQueue(cQ, D);