COSE213: Data Structure

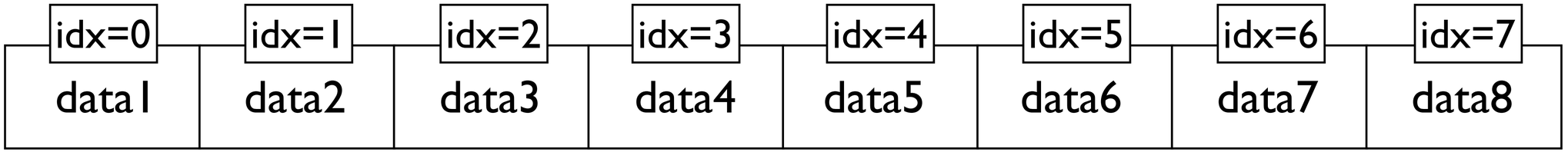
Lecture 4 - 큐(Queue)

Minseok Jeon

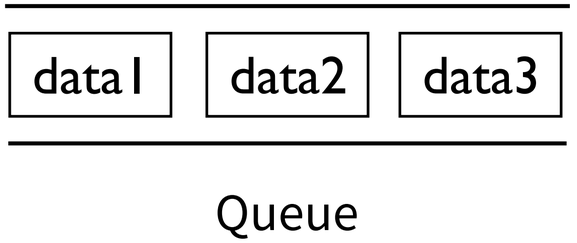
2024 Fall

리뷰: 배열 (Array)

- 배열(Array) 자료구조: 동일한 데이터 타입을 가진 값들을 연속된 공간에 저장하는 자료구조.



- 배열의 추상 자료형 (Abstract Data Type):
  - `create(type,size)` : 주어진 타입(`type`)과 길이(`size`)를 가지는 배열을 생성
  - `read(arr,index)` : 배열(arr)에서 주어진 인덱스(index)에 해당하는 자료를 반환
  - `update(arr,index,value)` : 배열(arr)에서 주어진 인덱스(index) 위치에 새로운 데이터(value)를 저장
- 배열은 다른 자료구조(e.g., 큐)들을 구현하는데 사용될 수 있음



문제: 수강신청 시스템

- 수강신청을 위한 접속자 관리
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 오래된(First) 데이터에 접근해야함

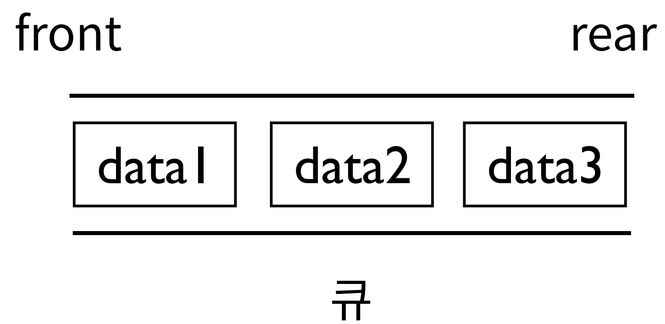
문제: 수강신청 시스템

- 수강신청을 위한 접속자 관리
  - 특징 1: 데이터간 (시간) 순서가 있음
  - 특징 2: 가장 오래된(First) 데이터에 접근해야함

- 필요한 자료구조의 형태: 대기줄 형태

해결책: 큐 (Queue)

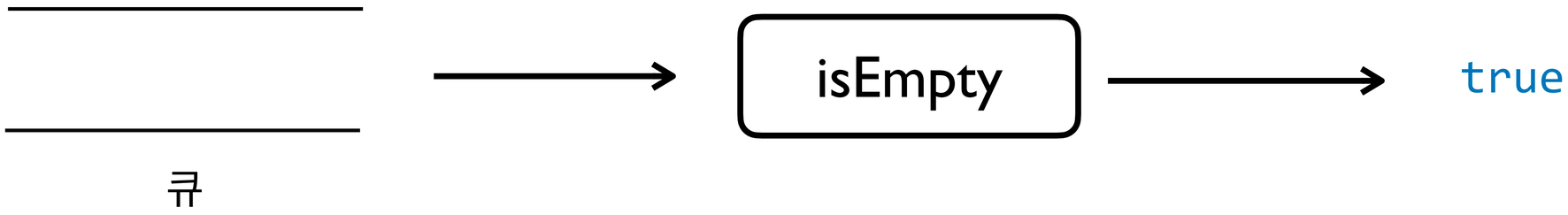
- 큐(Queue)는 선입선출(FIFO: Fist In, First Out) 원칙을 따르는 자료구조
- 큐의 추상 자료형:
  - create() : 비어있는 큐를 생성 후 반환
  - enqueue(q, e) : 큐 q에서 주어진 데이터 e를 큐의 맨 뒤에 추가
  - dequeue(q) : 큐 q가 비어있지 않으면 맨 앞 데이터를 삭제하고 반환
  - peek(q) : 큐 q가 비어있지 않으면 맨 앞 데이터를 제거하지 않고 반환
  - isEmpty(q) : 큐 q가 비어있으면 true를 아니면 false를 반환
  - isFull(q) : 큐 q가 가득 차 있으면 true를 아니면 false를 반환

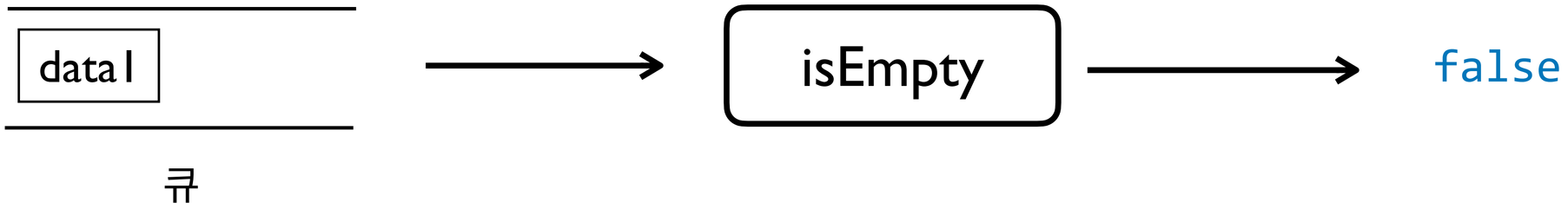


- ADT of array in Java: https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html
  ADT of array in Ocaml: https://ocaml.org/manual/5.2/api/Queue.html

isEmpty

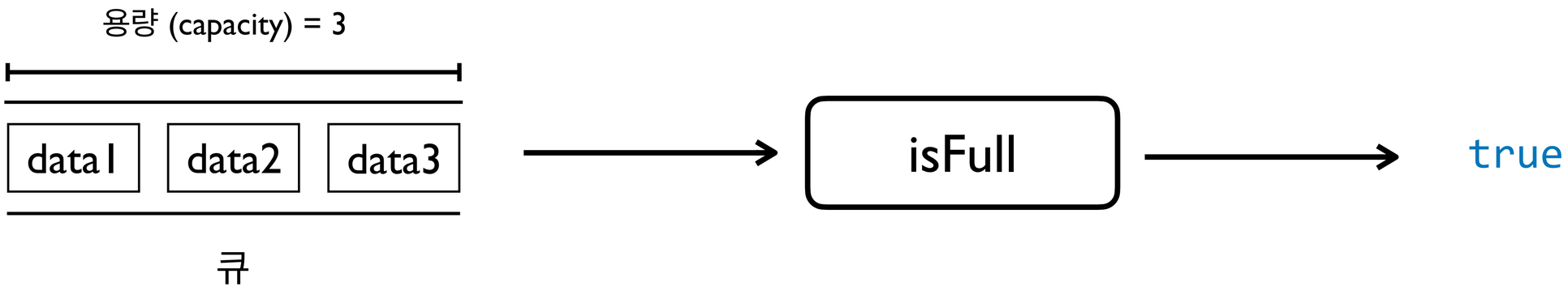
- `isEmpty` : 큐가 비어있으면 `true`를 아니면 `false`를 반환

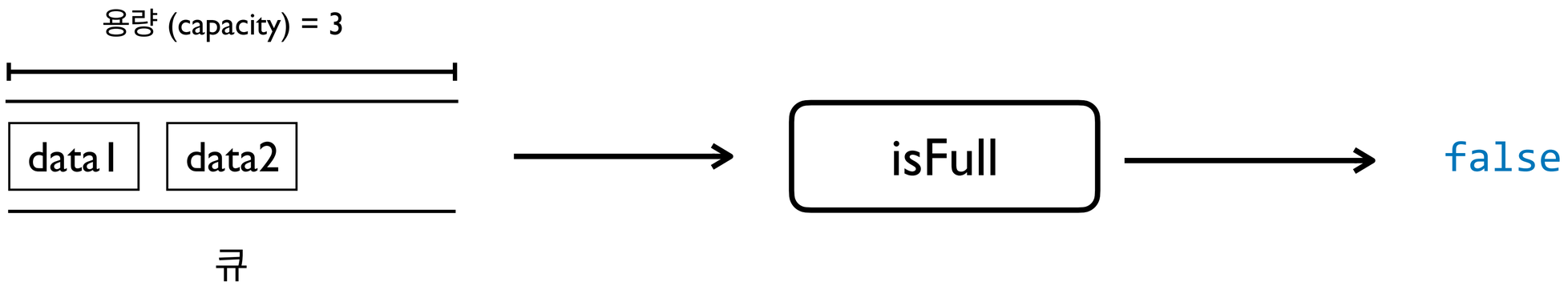




isFull

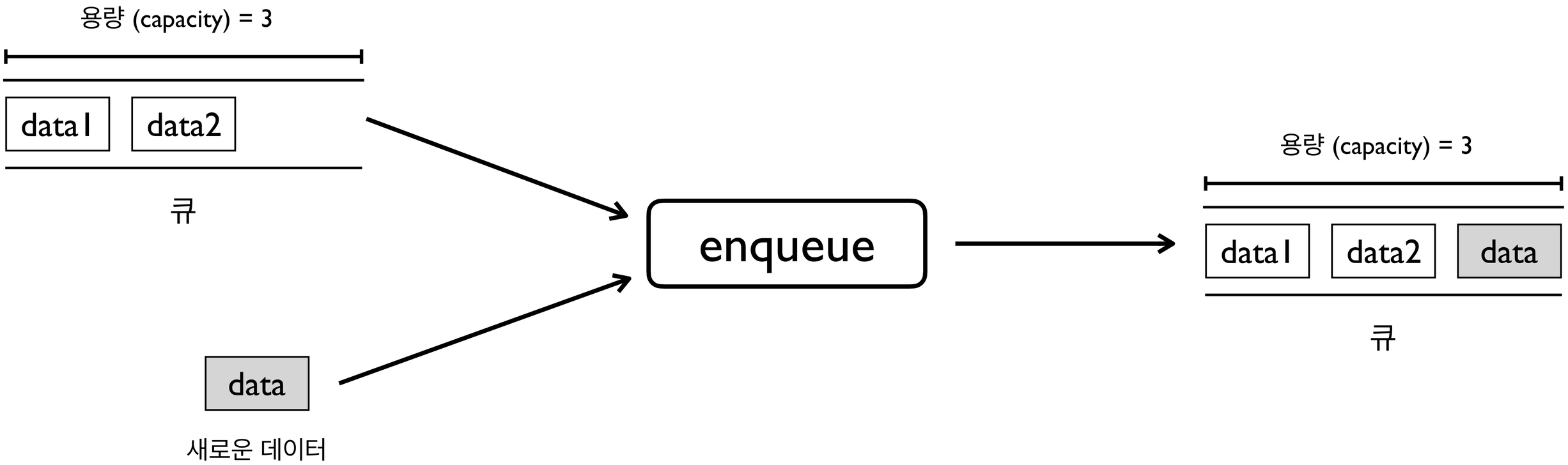
- `isFull` : 큐가 가득 차 있으면 `true`를 아니면 `false`를 반환





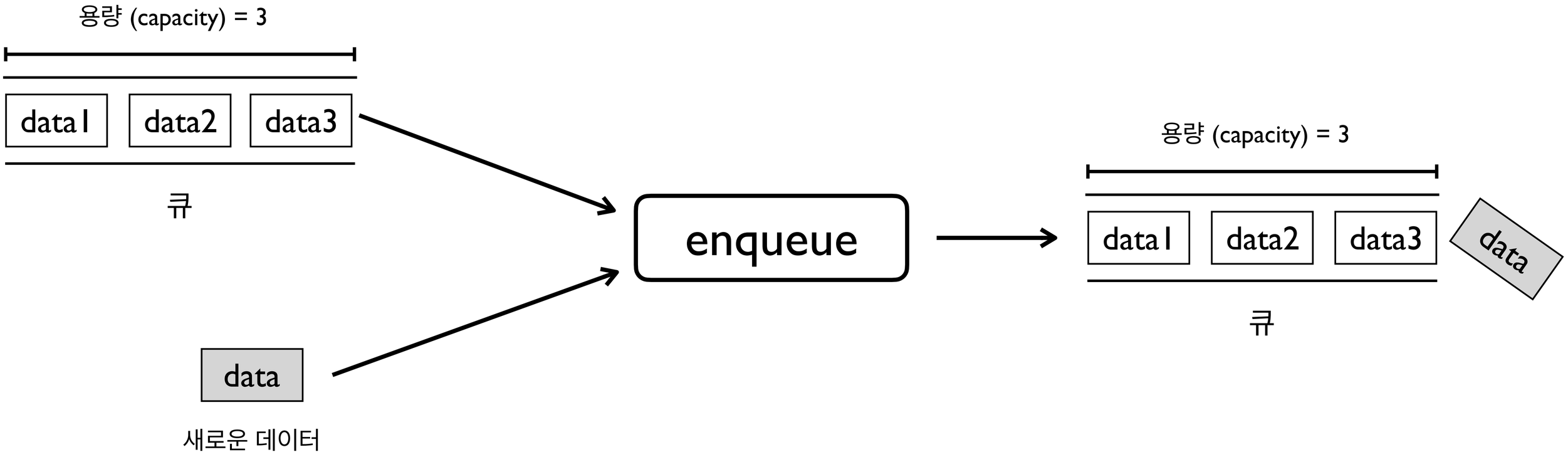
enqueue

- enqueue: 큐에서 주어진 데이터를 맨 뒤에 추가
  - 추가된 데이터가 큐의 가장 뒤(rear)에 위치하게 됨



enqueue

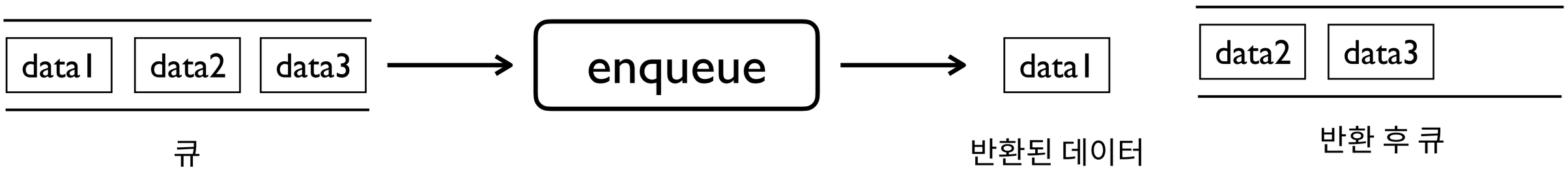
- enqueue: 큐에서 주어진 데이터를 맨 뒤에 추가
  - 추가된 데이터가 큐의 가장 뒤(rear)에 위치하게 됨
  - 추가할 공간이 없을 때 데이터를 추가할 경우 overflow 발생



dequeue

- dequeue : 큐가 비어있지 않으면 맨 앞 데이터를 삭제하고 반환

  - dequeue가 실행되기 전 앞(front)에서 두번째 데이터가 dequeue가 실행된 후 가장 앞에 위치하게 됨

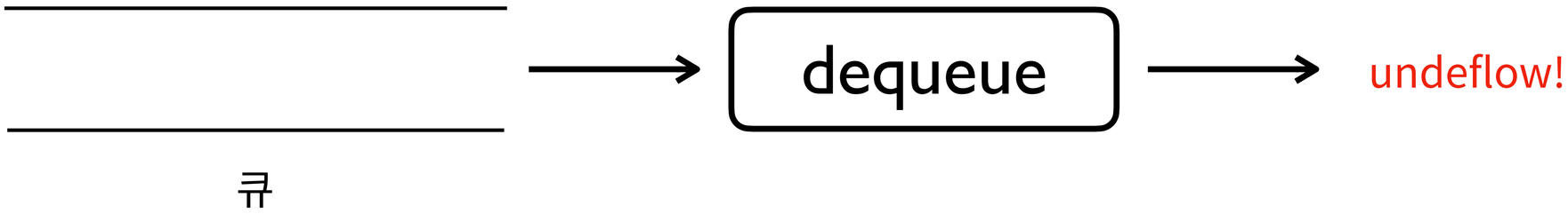


dequeue

- dequeue : 큐가 비어있지 않으면 맨 앞 데이터를 삭제하고 반환

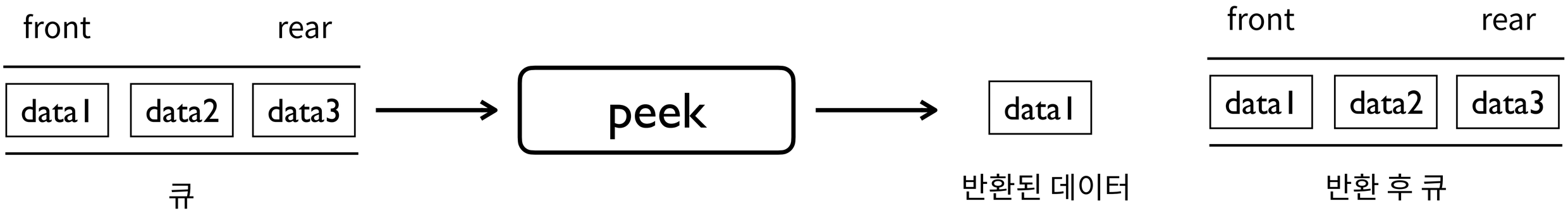
  - dequeue가 실행되기 전 앞(front)에서 두번째 데이터가 dequeue가 실행된 후 가장 앞에 위치하게 됨

  - 비어있는 큐에서 dequeue를 실행 할 경우 underflow

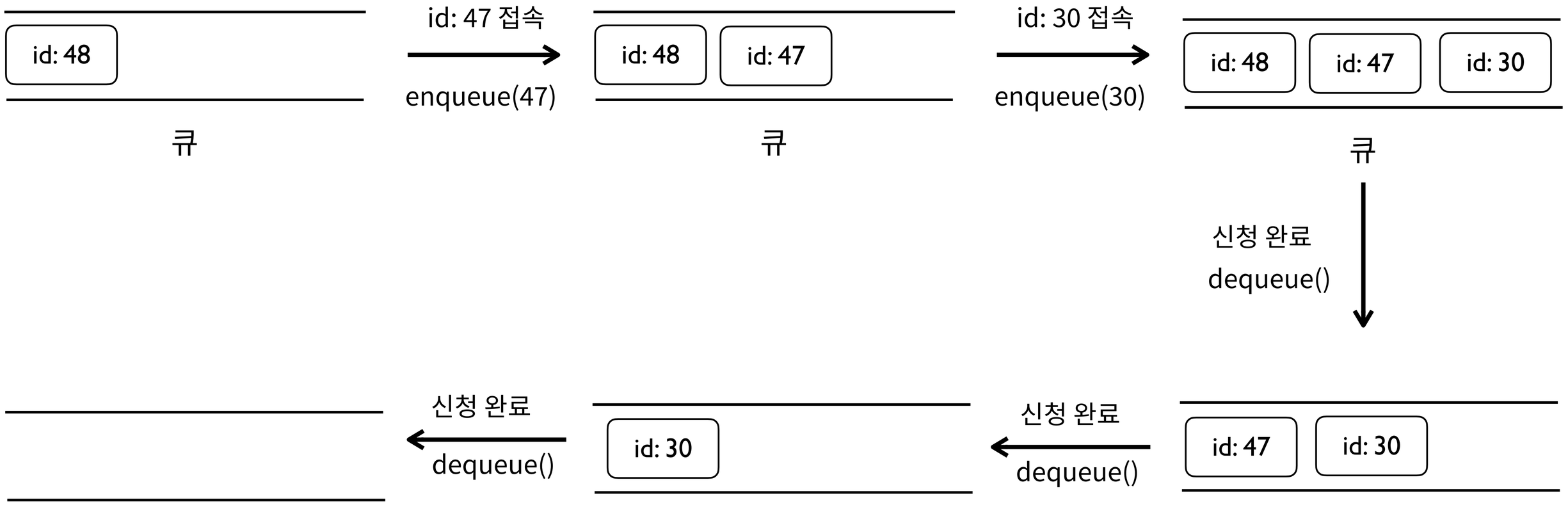


peek

- peek : 큐의 맨 앞 항목을 제거하지 않고 반환
  - Peek 실행 전후로 스택의 상태는 변하지 않음



수강신청 시스템


id: 48
큐
id: 47 접속
enqueue(47)
id: 48
id: 47
큐
id: 30 접속
enqueue(30)
id: 48
id: 47
id: 30
큐
신청 완료
dequeue()
id: 47
id: 30
신청 완료
dequeue()
id: 30
신청 완료
dequeue()

배열 큐 (Array Queue)

- 배열 큐의 추상 자료형:
  - create : 비어있는 배열 큐를 생성 후 포인터를 반환
  - enqueue : 큐의 맨 뒤에 주어진 새로운 정수 데이터를 추가
  - dequeue : 큐의 맨 앞에 있는 데이터를 삭제하고 반환
  - peek : 큐의 맨 앞에 있는 데이터을 제거하지 않고 반환
  - isEmpty : 큐가 비어있으면 `true`를 아니면 `false`를 반환
  - isFull : 큐가 가득 차 있으면 `true`를 아니면 `false`를 반환
  - size : 큐가 가지고 있는 데이터의 개수를 반환
  - destroy : 큐가 차지하고 있는 메모리를 해제함

- 배열 큐의 프로토타입:

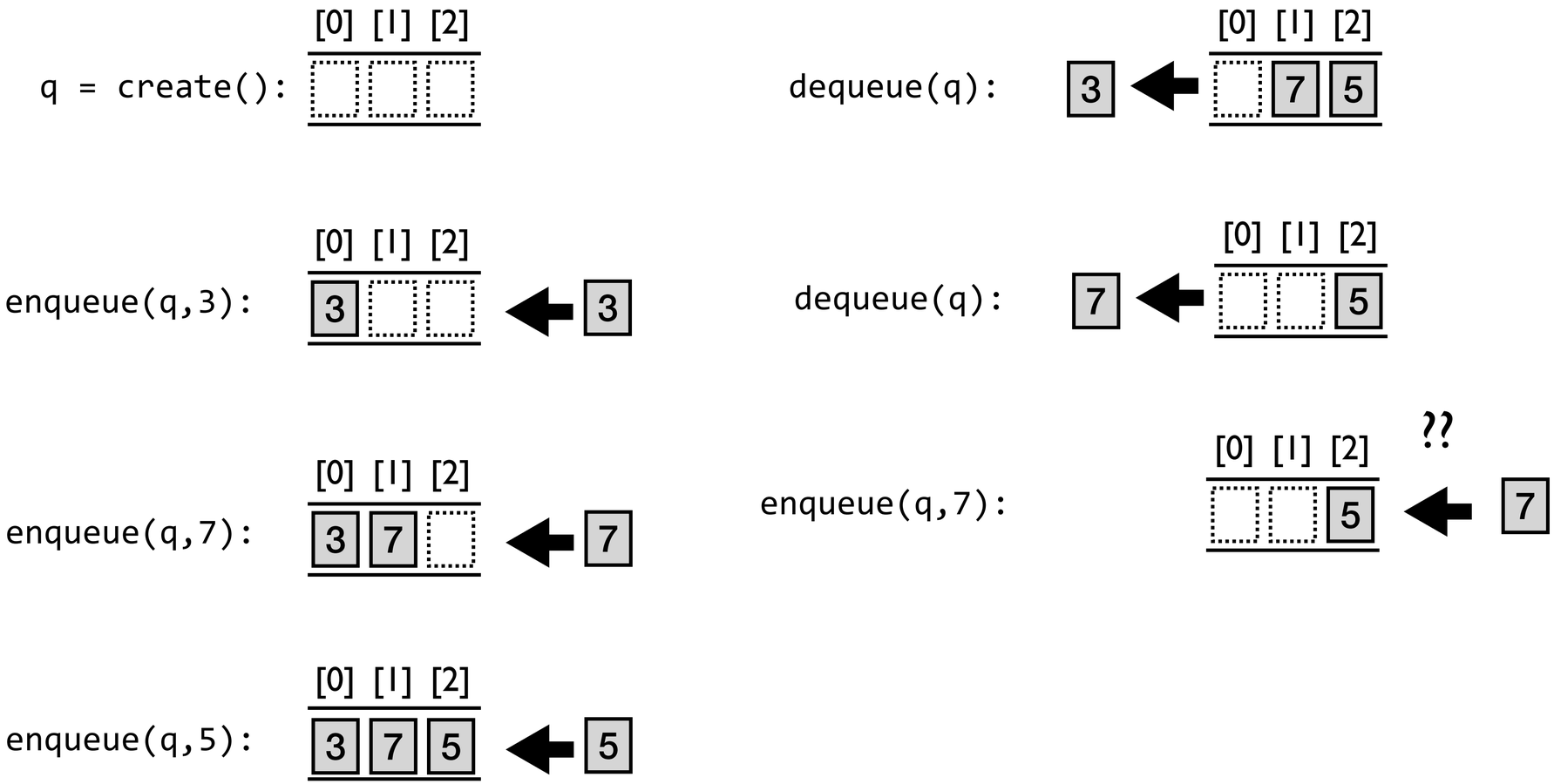
```
Queue* create();

void enqueue(Queue* s, int item);

int dequeue(Queue* s);

int peek(Queue* s);

bool isEmpty(Queue* s);

bool isFull(Queue* s);

int size(Queue* s);

void destroy(Queue* s);
```

Example

```c
#include <stdio.h>
#include "Queue.h"

int main() {
    Queue *q = create();

    enqueue(q, 3);
    enqueue(q, 7);
    enqueue(q, 5);


    printf("Dequeued element: %d\n", dequeue(q));

    printf("Front element: %d\n", peek(q));
    destroy(q);
    return 0;
}
```

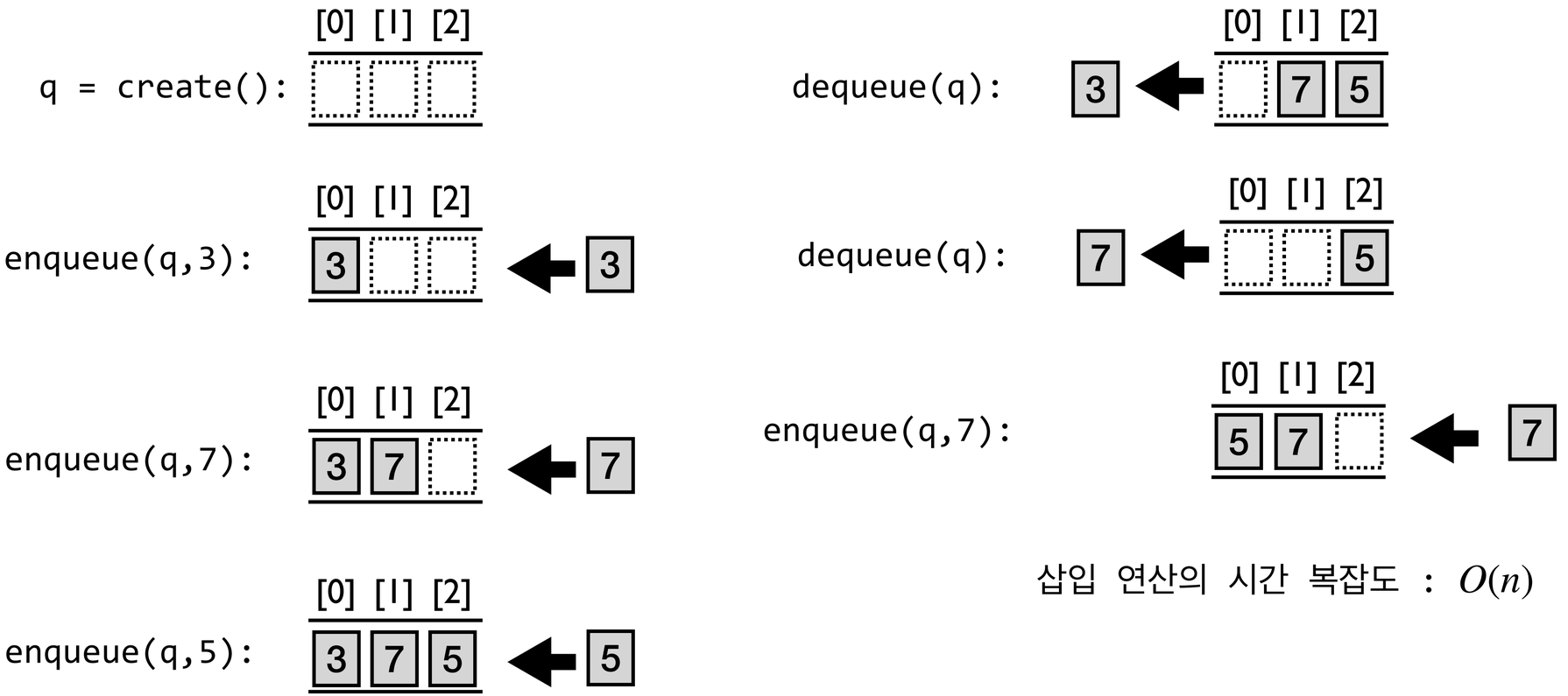
배열을 이용한 큐의 구현

- 선형 큐 (linear queue): 1차원 배열을 이용하여 큐를 구현하는 경우



배열을 이용한 큐의 구현

- 선형 큐 (linear queue): 1차원 배열을 이용하여 큐를 구현하는 경우
  - 선형 큐에서는 삽입에 추가적인 이동이 필요할 수 있음

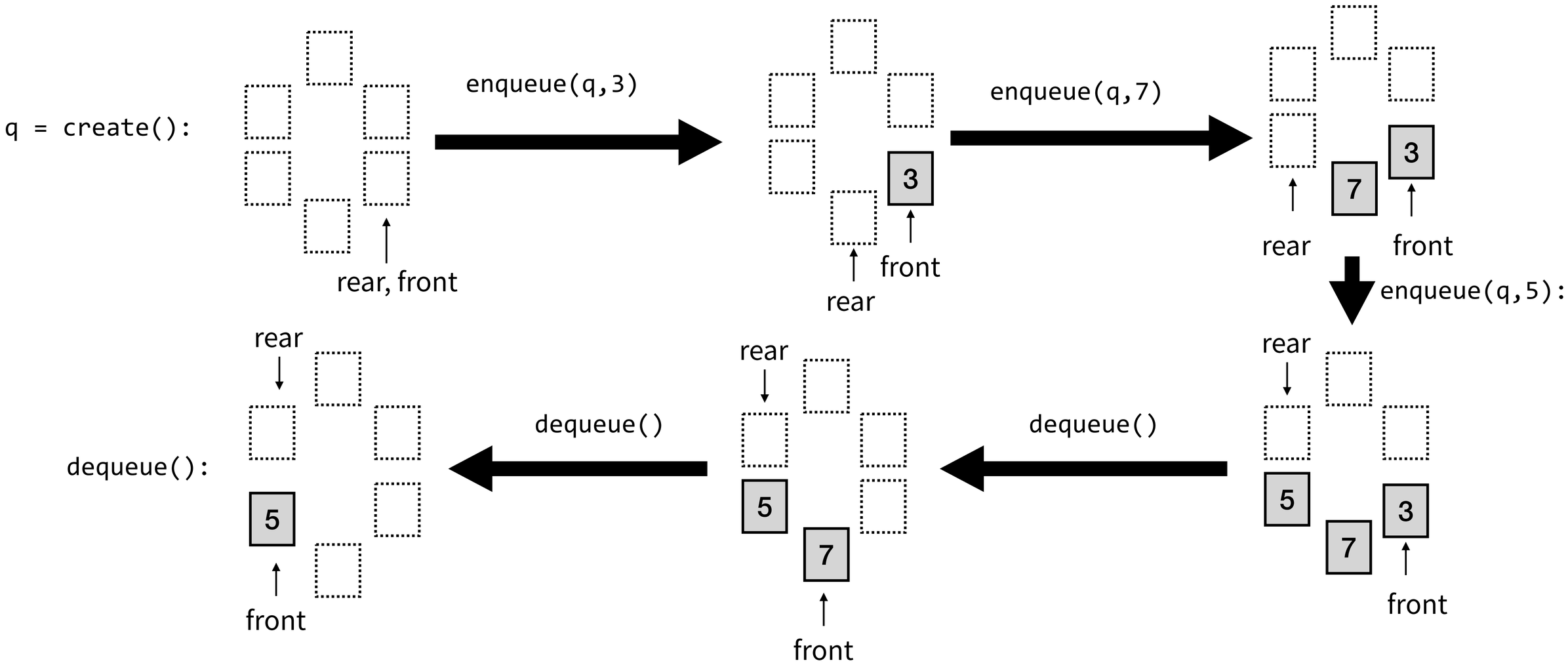


삽입 연산의 시간 복잡도 : $O(n)$

배열을 이용한 큐의 구현

rear: 저장하는 위치
front: 꺼내는 위치

- 해결책: 원형 큐 (배열을 원형으로 생각하기)



배열 큐 (Array Queue)

- 배열 큐(Array Queue)은 다음과 같은 정보를 가지는 자료구조임

```
typedef struct {
    int *items;
    int front;
    int rear;
    int size;
    int capacity;
} Queue;
```

배열 큐 (Array Queue)

- create : 비어있는 큐를 생성 후 반환

```
procedure create()
    queue ← allocateQueue()
    queue.items ← allocateArray()
    queue.front ← 0
    queue.rear ← 0
    queue.size ← 0
    queue.capacity ← maxCapacity()
    return queue
end procedure
```

슈도코드

```c
Queue create() {



}
```

C 코드 구현

배열 큐 (Array Queue)

- `isEmpty` : 큐가 비어있으면 `true`를 아니면 `false`를 반환

```
procedure isEmpty(queue)
    if queue.size = 0 then
        return true
    else
        return false
    end if
end procedure
```

```
bool isEmpty(Queue* q) {



}
```

- `isFull` : 큐가 가득 차 있으면 `true`를 아니면 `false`를 반환

```
procedure isFull(queue)
    if queue.size = queue.capacity then
        return true
    else
        return false
    end if
end procedure
```

```
bool isFull(Queue* q) {



}
```

배열 큐 (Array Queue)

- enqueue : 큐의 맨 뒤에 주어진 새로운 정수 데이터를 추가

```
procedure enqueue(queue, data)
  if isFull(queue) then
    print("Cannot enqueue. Queue is full.")
    return error()                              ▷ failed
  end if
  idx ← queue.rear
  queue.items[idx] ← data
  queue.rear ← (queue.rear + 1) mod queue.capacity
  queue.size ← queue.size + 1
  return queue
end procedure
```

```c
void enqueue(Queue *q, int value) {









}
```

배열 큐 (Array Queue)

- dequeue : 큐의 가장 앞에 있는 데이터를 삭제하고 반환

```
procedure dequeue(queue)
  if isEmpty(queue) then
    print("Cannot dequeue. Queue is empty.")
    return error()                                  ▷ failed
  end if
  item ← queue.items[queue.front]
  queue.size  ← queue.size - 1
  queue.front ← (queue.front + 1) mod queue.capacity
  return item                                       ▷ succeed
end procedure
```

```
int dequeue(Queue *q) {




}
```

배열 큐 (Array Queue)

- peek : 큐의 맨 앞에 있는 데이터을 제거하지 않고 반환

```
procedure peek(queue)
  if isEmpty(queue) then
    print("Cannot peek. Queue is empty.")
    return error()                          ▷ failed
 return queue.items[queue.front]            ▷ succeed
end procedure
```

```c
int peek(Queue *q) {




}
```

배열 큐 (Array Queue)

- destroy : 큐가 차지하고 있는 메모리를 해제함

```
procedure destroy(queue)
  free(queue.items)
  free(queue)
end procedure
```

```
void destroy(Queue *q) {



}
```

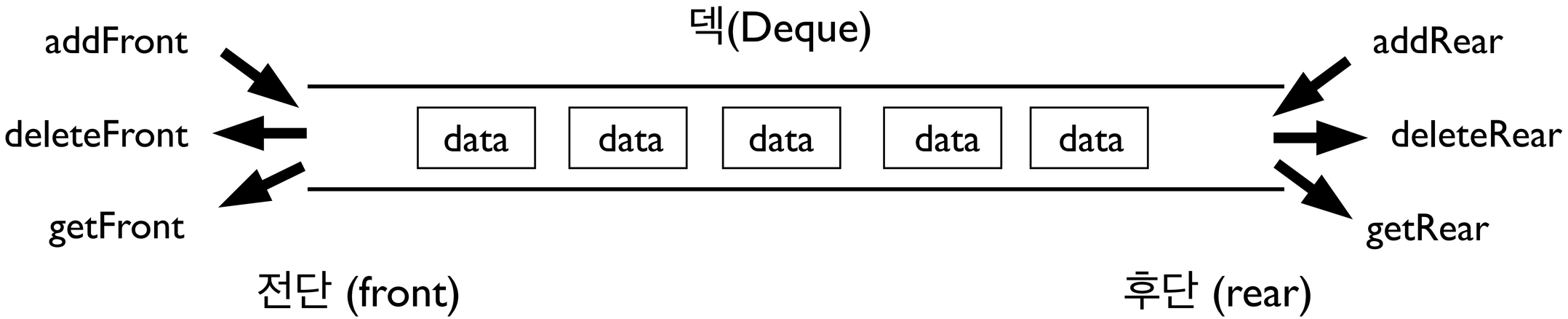
- size : 큐가 가지고 있는 데이터의 개수를 반환

```
procedure size(queue)
   return queue.size
end procedure
```

```
int size(Queue *q) {



}
```

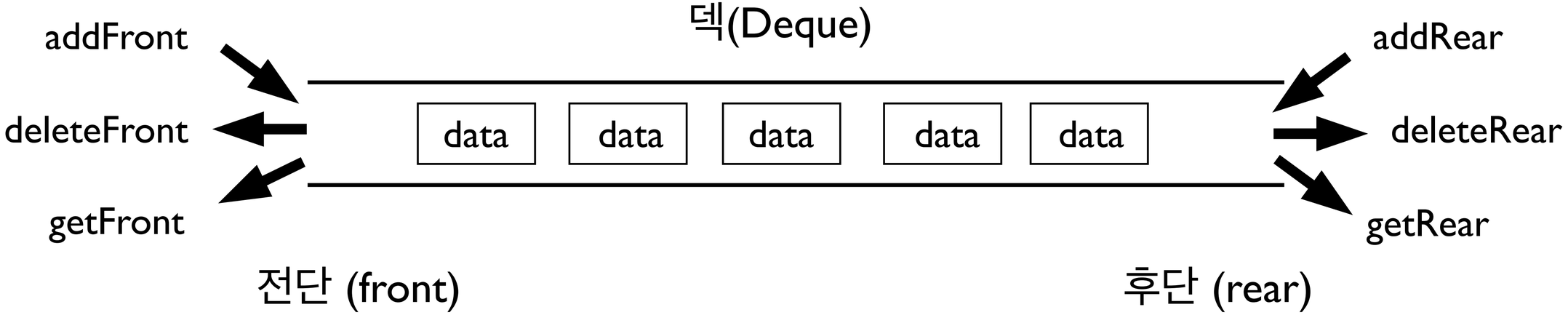
덱 (Deque: double-ended queue)

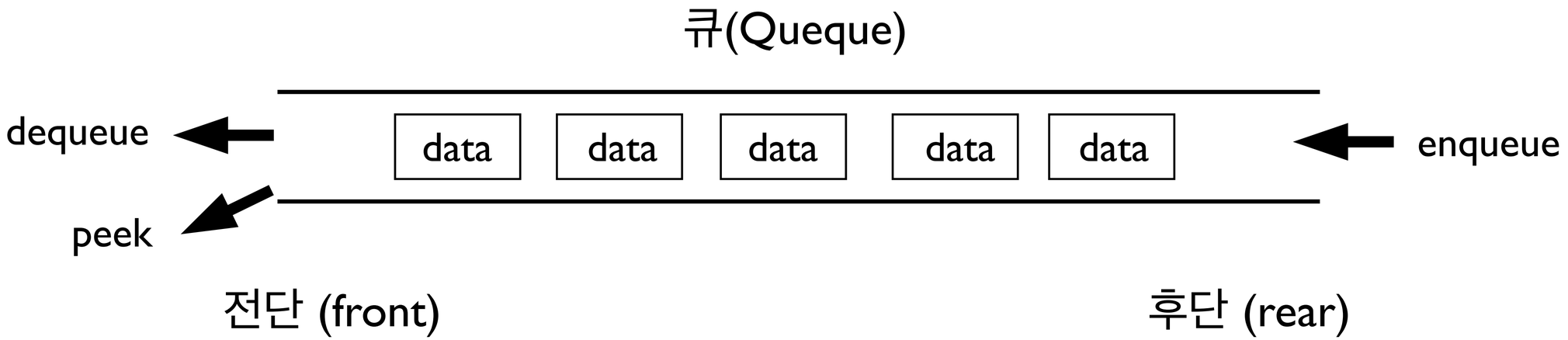
- 덱 (Deque): 전단(front)과 후단(rear)에 모두 삽입과 삭제가 가능한 큐



덱 (Deque: double-ended queue)

- 덱 (Deque): 전단(front)과 후단(rear)에 모두 삽입과 삭제가 가능한 큐





Example: 회문 (Palindrome) 검사

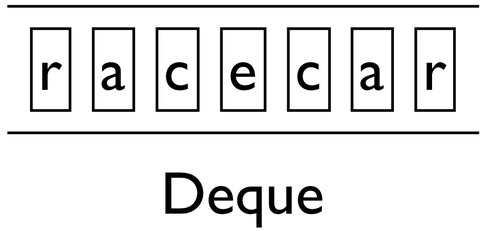
- 회문: 앞에서 읽으나 뒤에서 읽으나 동일한 문자열
  - 회문 예시: “radar”, “level”, “madam”, “racecar”, “121”, “1331”, “12321”
- 공백, 대소문자, 구두점등을 무시하고 검사하는 경우도 있음

"A man, a plan, a canal, Panama!"

Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

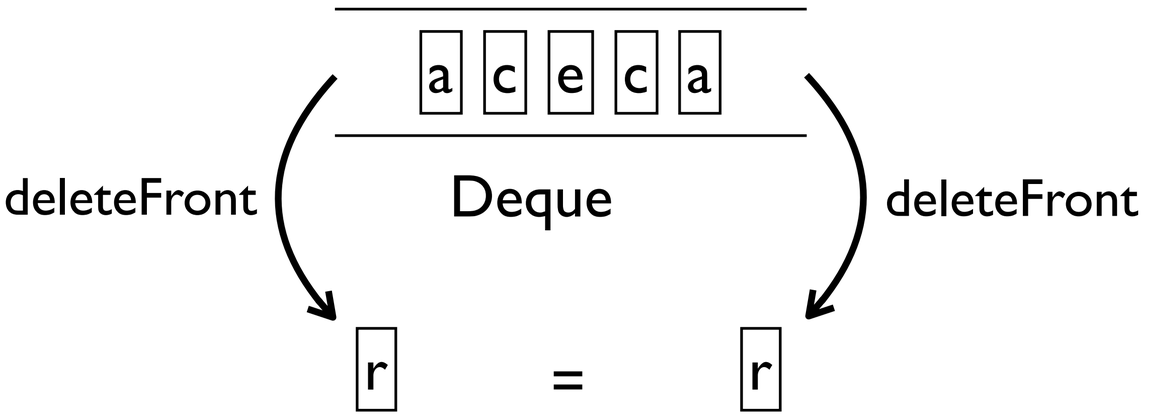


Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

  (2) deleteFront와 deleteRear가 같은 값을 반환하는지 확인함



Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

  (2) deleteFront와 deleteRear가 같은 값을 반환하는지 확인함

  (3) 다르다면 false를 반환함 (회문 아님). 같다면 덱에 1개 또는 0개의 데이터가 남을 때까지 (2)를 반복

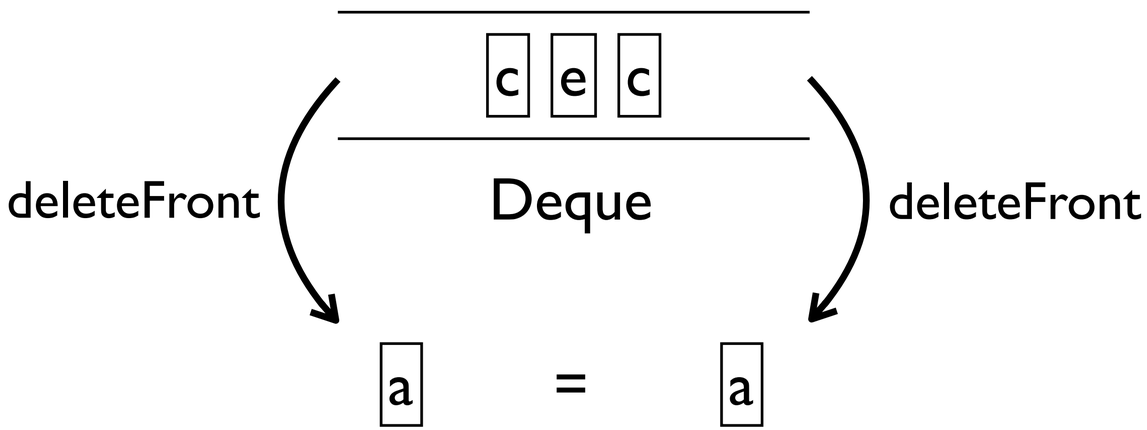
Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

  (2) deleteFront와 deleteRear가 같은 값을 반환하는지 확인함

  (3) 다르다면 false를 반환함 (회문 아님). 같다면 덱에 1개 또는 0개의 데이터가 남을 때까지 (2)를 반복



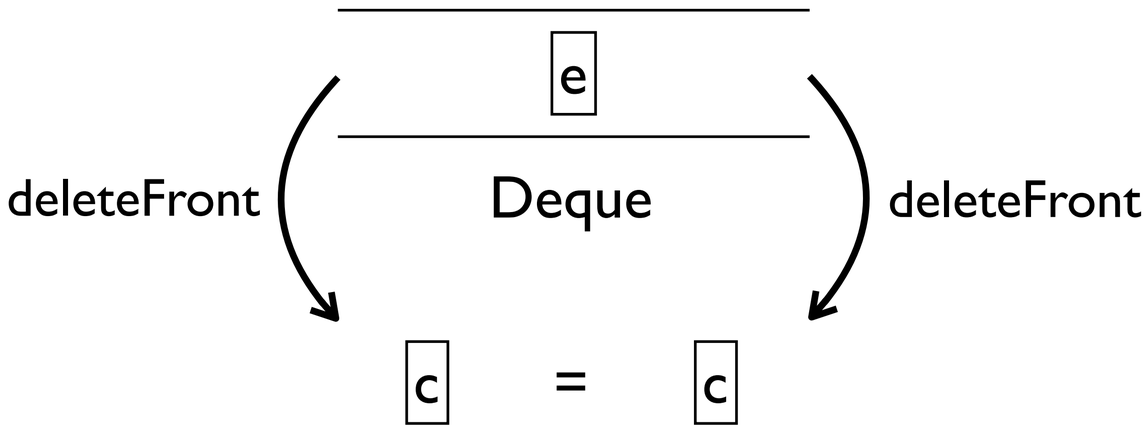
Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

  (2) deleteFront와 deleteRear가 같은 값을 반환하는지 확인함

  (3) 다르다면 false를 반환함 (회문 아님). 같다면 덱에 1개 또는 0개의 데이터가 남을 때까지 (2)를 반복



Example: 회문 (Palindrome) 검사

- 덱을 이용한 회문 검사 알고리즘

  (1) 문자열(e.g., “racecar”)을 빈 덱에 추가함

  (2) deleteFront와 deleteRear가 같은 값을 반환하는지 확인함

  (3) 다르다면 false를 반환함 (회문 아님). 같다면 덱에 1개 또는 0개의 데이터가 남을 때까지 (2)를 반복

  (4) true를 반환함 (회문임).

덱 (Deque)

- 덱(Deque: double-ended queue)은 전단(front)과 후단(rear)에서 모두 삽입, 삭제, 접근이 가능한 큐
- 덱의 추상 자료형:
  - create : 비어있는 덱을 생성 후 반환
  - addFront : 덱의 맨 앞에 데이터를 추가
  - deleteFront : 큐의 맨 앞에 있는 데이터를 삭제하고 반환
  - getFront : 덱의 맨 앞에 있는 데이터을 제거하지 않고 반환
  - addRear : 덱의 맨 뒤에 데이터를 추가
  - deleteRear : 큐의 맨 뒤에 있는 데이터를 삭제하고 반환
  - getRear : 덱의 맨 뒤에 있는 데이터을 제거하지 않고 반환
  - isEmpty : 덱이 비어있으면 true를 아니면 false를 반환
  - isFull : 덱이 가득 차 있으면 true를 아니면 false를 반환
  - size : 덱에 들어있는 데이터의 개수를 반환
- 프로토타입:

```
Deque* create();
void addFront(Deque* dq, int value);
int deleteFront(Deque* dq);
int getFront(Deque* dq)
void addRear(Deque* dq, int value);
int deleteRear(Deque* dq)
int getRear(Deque* dq)
bool isEmpty(Deque* dq);
bool isFull(Deque* dq);
int size(Deque* dq);
```

Example: 회문(palindrome) 검사

```
#include “Deque.h"

bool isPalindrome(char* str) {
  int len = strlen(str);
  Deque *dq = create();
  for (int i = 0; i < len; i++) {
    addRear(dq, str[i]);
  }

  while (size(dq) > 1) {
    char frontChar = deleteFront(dq);
    char rearChar = deleteRear(dq);
    if (frontChar != rearChar) {
      return false;
    }
  }
  return true;
}
```

마무리 (Wrap-up)

- 문제:
  - 먼저 들어온 데이터를 먼저 처리해야 하는 상황(e.g., 대기열)에 적합한 자료구조가 필요함
- 해결책:
  - 큐(Queue): 선입선출(FIFO: First In, First Out) 원칙을 따르는 자료구조
  - 큐 자료구조는 다음의 기능들을 제공함 (큐의 추상 자료형):
    - create() : 비어있는 큐를 생성 후 반환
    - enqueue(q, e) : 큐 q에서 주어진 데이터 e를 큐의 맨 뒤에 추가
    - dequeue(q) : 큐 q가 비어있지 않으면 맨 앞 데이터를 삭제하고 반환
    - peek(q) : 큐 q가 비어있지 않으면 맨 앞 데이터를 제거하지 않고 반환
    - ...