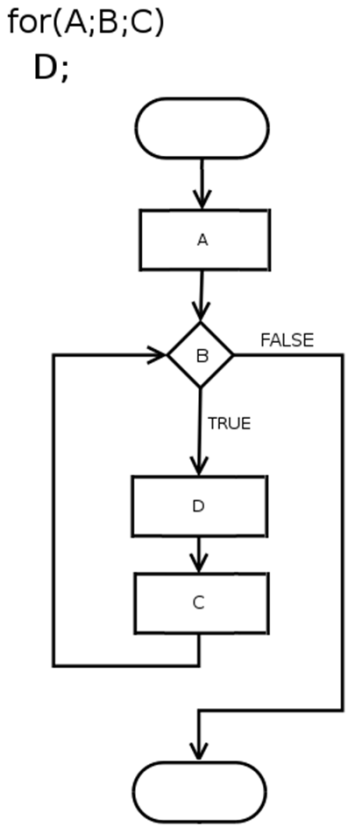
Clickseo.com

자료구조 & 알고리즘

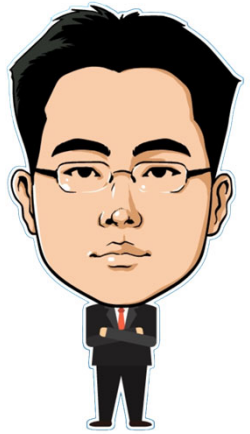


큐
(Queue)

Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



목 차

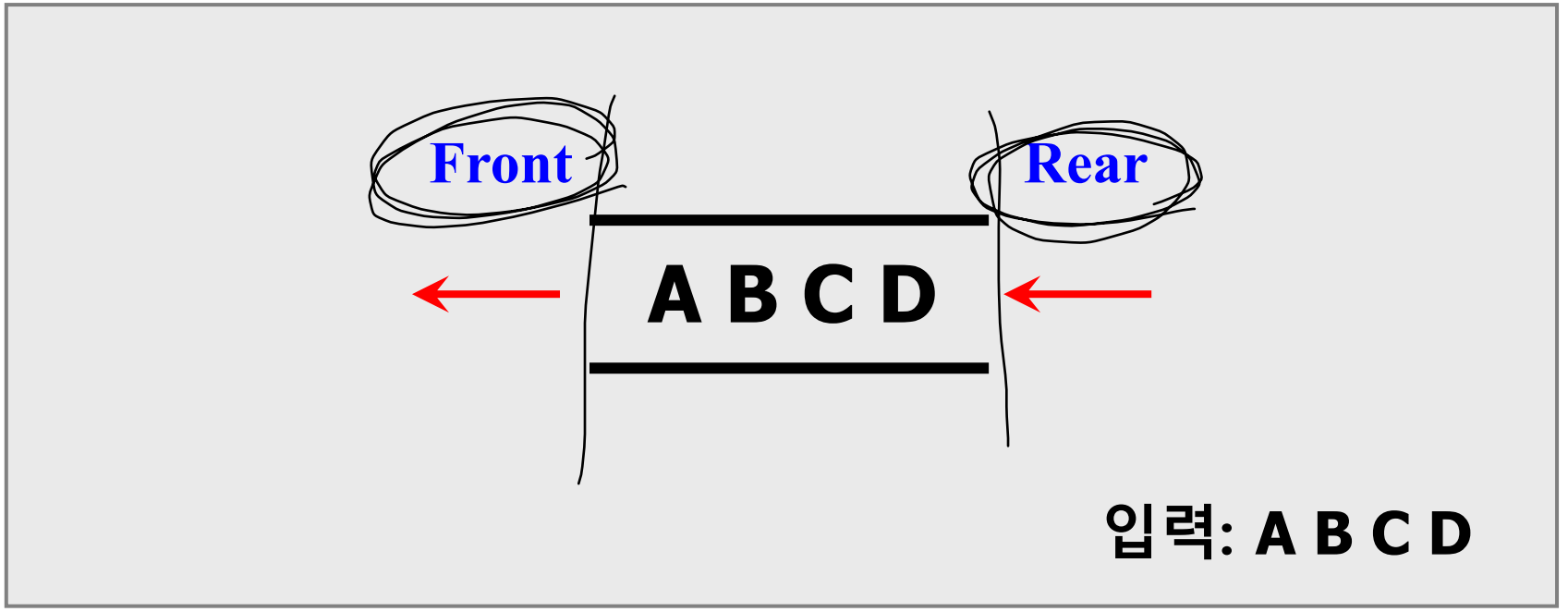
- 큐의 이해
- 큐 구현
- 큐 응용

큐의 이해

- **큐의 이해**
  - 선형 큐
  - 원형 큐
- **큐 구현**
- **큐 응용**

큐의 이해

- **큐(Queue)**
  - **선입선출**(FIFO, First-In First-Out)
    - 리스트의 한쪽 끝에서 삽입 작업이 이루어지고 반대쪽 끝에서 삭제 작업이 이루어져서 삽입된 순서대로 삭제되는 구조

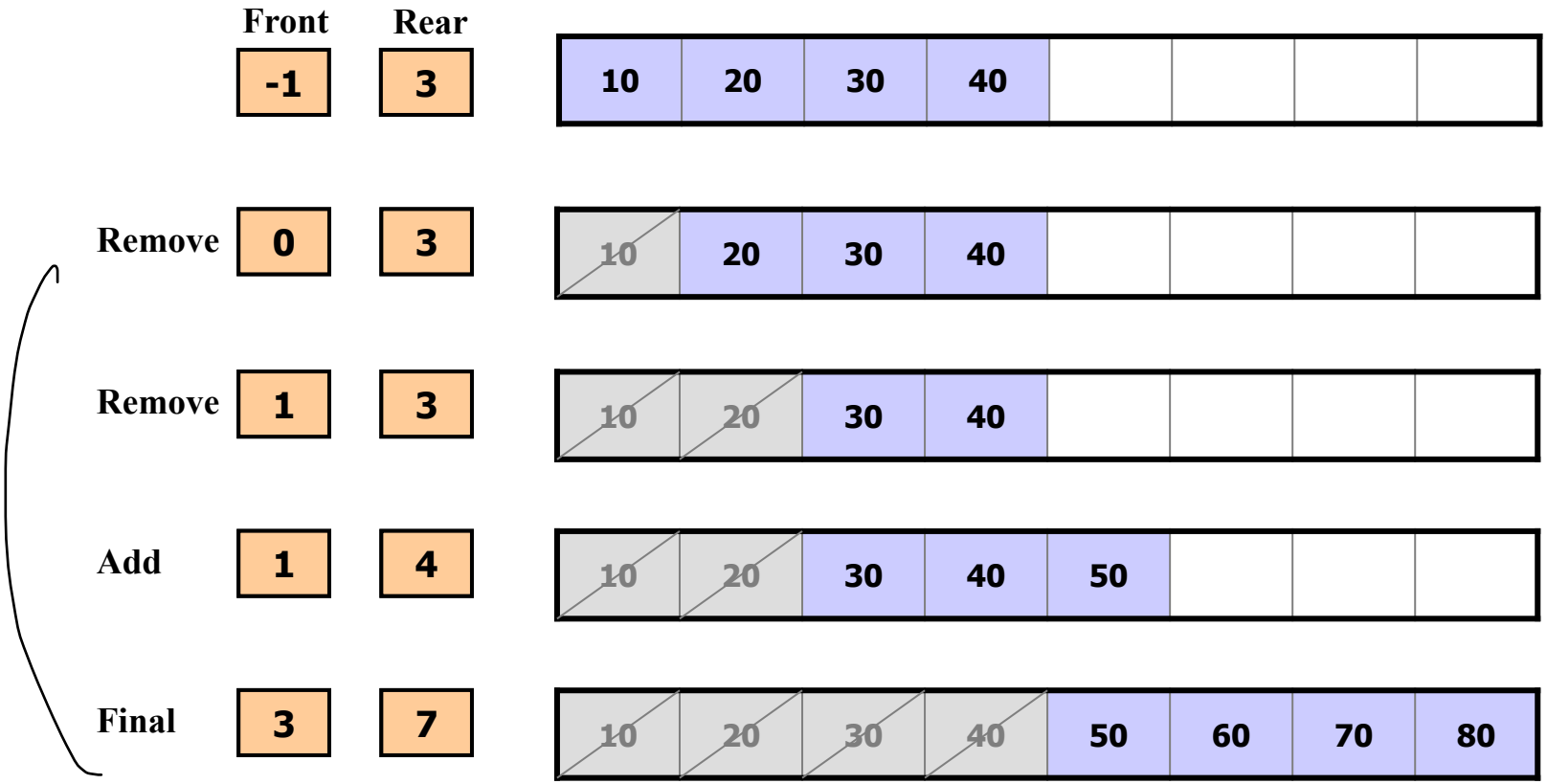


큐의 이해

선형 큐, 원형 큐

선형 큐 (1/2)

- **선형 큐(Linear Queue)**
  - 연속된 삽입, 삭제에 의한 오른쪽 이동

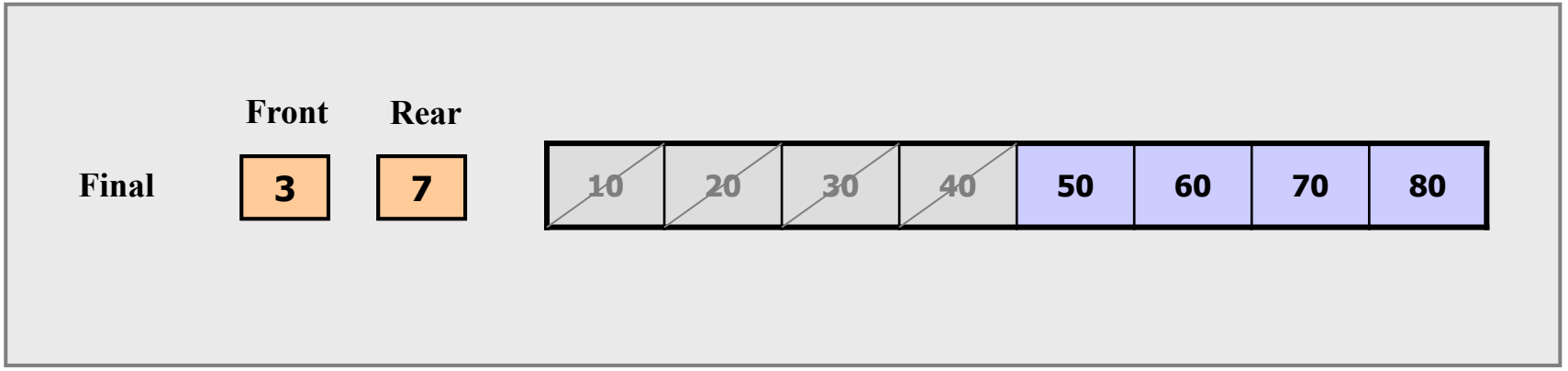


선형 큐 (2/2)

● 선형 큐

❍ 선형 큐의 잘못된 포화 상태 인식

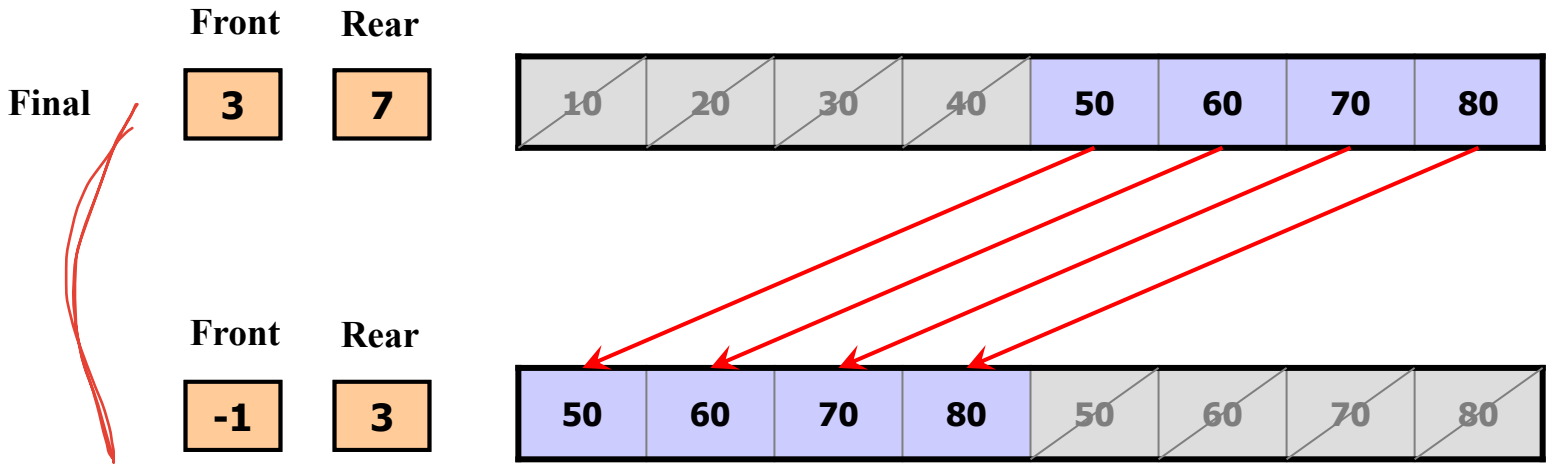
- 큐에서 삽입과 삭제를 반복하면서 아래와 같은 상태일 경우, 앞부분에 빈자리가 있지만 **rear = n - 1 상태이므로 포화 상태로 인식하고** 더 이상의 삽입을 수행하지 않는다.



Rear + 1 == Queue_SIZE

원형 큐 (1/3)

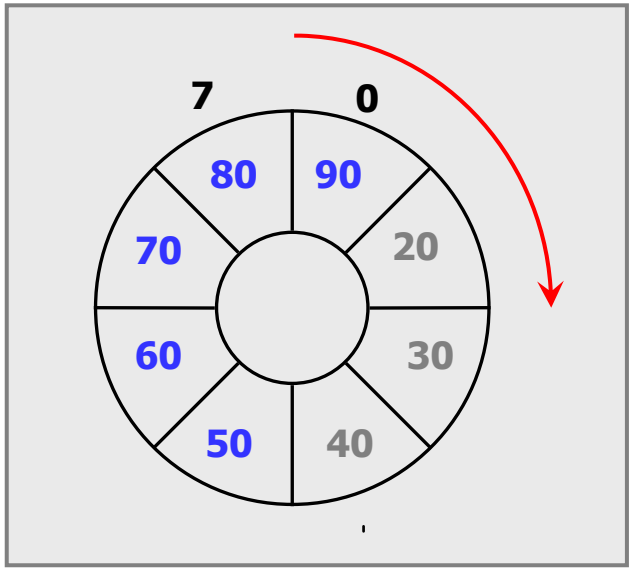
- **선형 큐**
  - 선형 큐의 잘못된 포화상태 인식의 해결 방법 #1
    - 저장된 원소들을 배열의 앞부분으로 이동
      - 순차자료에서의 이동 작업은 연산이 복잡하여 효율성이 떨어짐

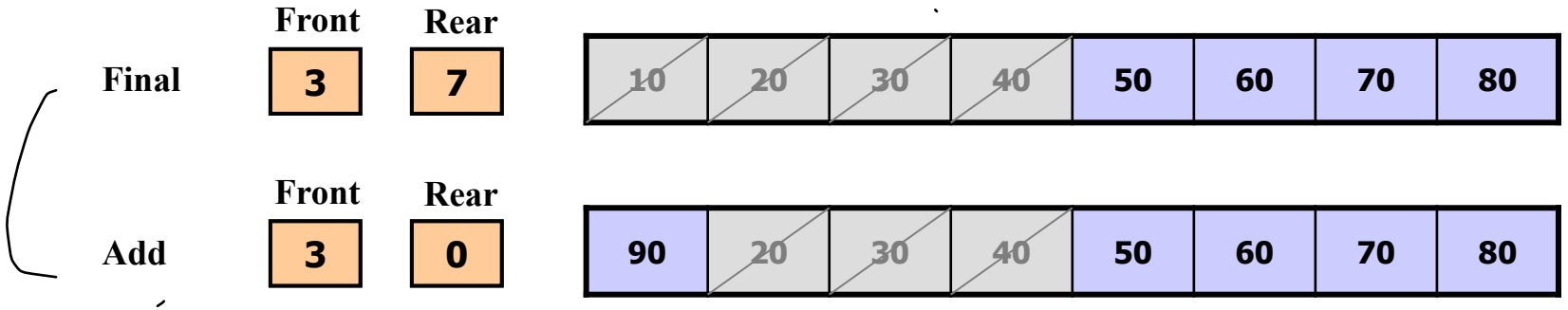


원형 큐 (2/3)

- **원형 큐**(Circular Queue)
  - 선형 큐의 잘못된 포화상태 인식의 해결 방법 #2

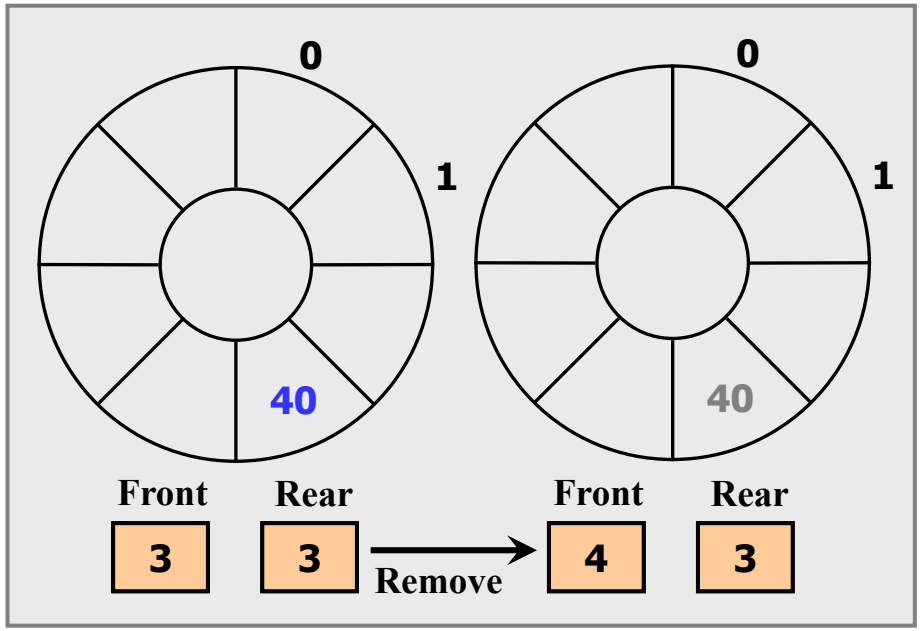
(Rear + 1) % Queue_SIZE

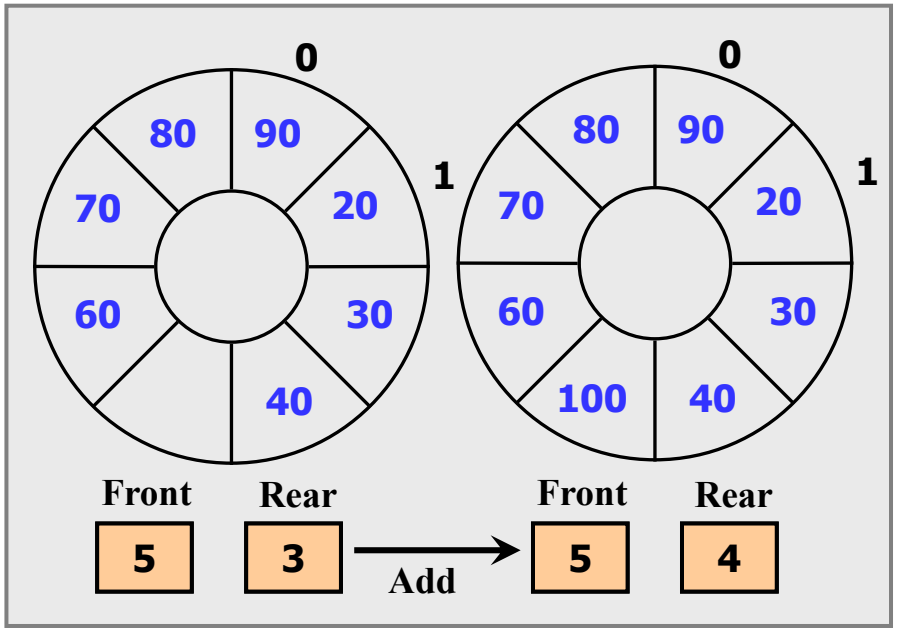




원형 큐 (3/3)

- **원형 큐:** 문제점
  - 문제점
    - 빈 큐와 꽉 찬 큐의 판정불가: Front = Rear + 1
    - 별도의 Count 변수 유지





큐 구현

- 큐의 이해
- **큐 구현**
  - 순차 자료구조
  - 연결 자료구조
- 큐 응용

큐 구현: 알고리즘 (1/4)

● 선형 큐 구현: 알고리즘

❍ 초기 공백 선형 큐 생성 알고리즘

```
queue_Create()
    Q[n];              // 크기가 n 인 1차원 배열 생성
    front ← -1;
    rear ← -1;
end queue_Create()
```

- 큐에 최대로 저장할 수 있는 원소 개수를 배열의 크기로 하는 1차원 배열을 선언
  - 저장된 원소가 없는 공백 큐이므로 front와 rear는 -1로 초기화

큐 구현: 알고리즘 (2/4)

- **선형 큐 구현:** 알고리즘
  - 선형 큐의 삽입 알고리즘: **enQueue**

```
enQueue(Q, data)
    if (Q->rear + 1 = n) then  queue_isFull;
    else
          Q[++rear] ← data;
end enQueue()
```

  - 선형 큐의 삭제 알고리즘 : **deQueue**

```
deQueue(Q)
    if (Q->front = Q->rear) then  queue_isEmpty;
    else
          return Q[++front];
end deQueue()
```

큐 구현: 알고리즘 (3/4)

- **선형 큐 구현:** 알고리즘
  - 선형 큐의 공백 검사 알고리즘

```
queue_isEmpty(Q)
    if(Q->front = Q->rear) then return true;
    else return false;
end queue_isEmpty()
```

  - 선형 큐의 포화 상태 검사 알고리즘

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

큐 구현: 알고리즘 (4/4)

- **선형 큐 구현: 알고리즘**
  - 선형 큐 검색 알고리즘

```
queue_Peek()
    if(Q->front = Q->rear) then  queue_isFull;
    else
          return Q[front + 1];
end queue_Peek()
```

  - 큐에서 검색은 원소 중에서 가장 먼저 들어와 있는 원소, 즉 Q[front+1]에 있는 원소를 검색하여 반환하는 연산이다.

큐 구현

순차 자료구조

큐 구현: 순차 자료 구조 (1/3)

- **큐 구현:** 순차 자료구조

C

```c
#define queueMAXSIZE  100

typedef     int         element;
typedef  struct  _arrayQueue {
     element            queue[queueMAXSIZE];
     int                front, rear;
}arrayQueue;

arrayQueue*             queueCreate(void);
void                    queueDestroy(arrayQueue*);
void                    enQueue(arrayQueue* Q, element item);
element                 deQueue(arrayQueue* Q);
element                 peek(arrayQueue* Q);
int                     isEmpty(arrayQueue* Q);
int                     isFull(arrayQueue* Q);
void                    queuePrint(arrayQueue* Q);
```

큐 구현: 순차 자료 구조 (2/3)

- **큐 구현:** 순차 자료구조

```
#define queueMAXSIZE  100

template <typename E>
class arrayQueue {
private:
    E       queue[queueMAXSIZE];
    int     front, rear;
public:
    arrayQueue();
    ~arrayQueue();
    void    enQueue(const E& e);
    E       deQueue(void);
    E       peek(void) const;
    bool    isEmpty(void) const;
    bool    isFull(void) const;
    void    printQueue(void) const;
};
```

C++

큐 구현: 순차 자료 구조 (3/3)

- **큐 구현:** 순차 자료구조

```
class ListQueue:
    def __init__(self):
        self.__queue = []
    def enQueue(self, num):
    def deQueue(self):
    def peek(self):
    def isEmpty(self) -> bool:
    def dequeueAll(self):
    def printQueue(self):
```

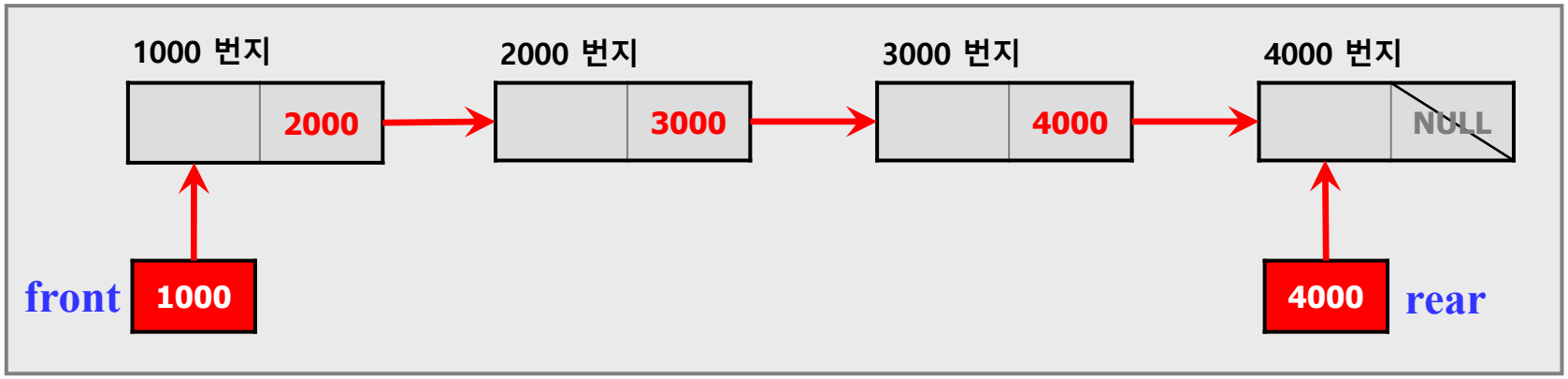
Python

큐 구현

연결 자료구조

큐 구현: 연결 자료 구조 (1/6)

- **연결 큐(Linked Queue)**
  - **순자 자료구조로 구현한 큐의 문제점**
    - 크기가 제한되어 큐의 길이를 마음대로 사용할 수 없다.
    - 원소가 없을 때에도 항상 그 크기를 유지하고 있어야 하므로 메모리 낭비
  - **연결 큐의 구조**
    - **데이터 필드와 링크 필드를 가진 노드로 구성**
      - **front** : 첫 번째 노드를 가리키는 포인터
      - **rear** : 마지막 노드를 가리키는 포인터
      - **초기 상태(공백 큐): front와 rear 모두 널(NULL) 포인터로 설정**



큐 구현: 연결 자료 구조 (2/6)

- **큐 구현:** 연결 자료구조

C

```c
#define queueMAXSIZE  100

typedef int element;
typedef struct _queueNode {
    element                 data;
    struct _queueNode*      link;
}queueNode;

typedef struct _LinkedQueue {
    queueNode* front, *rear;
}LinkedQueue;

LinkedQueue*        queueCreate(void);
void                queueDestroy(LinkedQueue* q);
queueNode*          makeQueueNode(int data);
void                enQueue(LinkedQueue* q, element data);
element             deQueue(LinkedQueue* q);
element             peek(LinkedQueue* q);
_Bool               isEmpty(LinkedQueue* q);
void                printQueue(LinkedQueue* q);
```

큐 구현: 연결 자료 구조 (3/6)

- **큐 구현:** 연결 자료구조

```
template <typename E>
class queueNode {
private:
      E                    data;
      queueNode<E>*        link;
      template <typename E> friend class LinkedQueue;
};

template <typename E>
class LinkedQueue {
private:
      queueNode<E>* front, *rear;
public:
      LinkedQueue();
      ~LinkedQueue();
      queueNode<E>* makeQueueNode(const int& num) const;
      void                 enQueue(const E& e);
      E                    deQueue(void);
      E                    peek(void) const;
      bool                 isEmpty(void) const;
      void                 printQueue(void) const;
};
```

C++

큐 구현: 연결 자료 구조 (4/6)

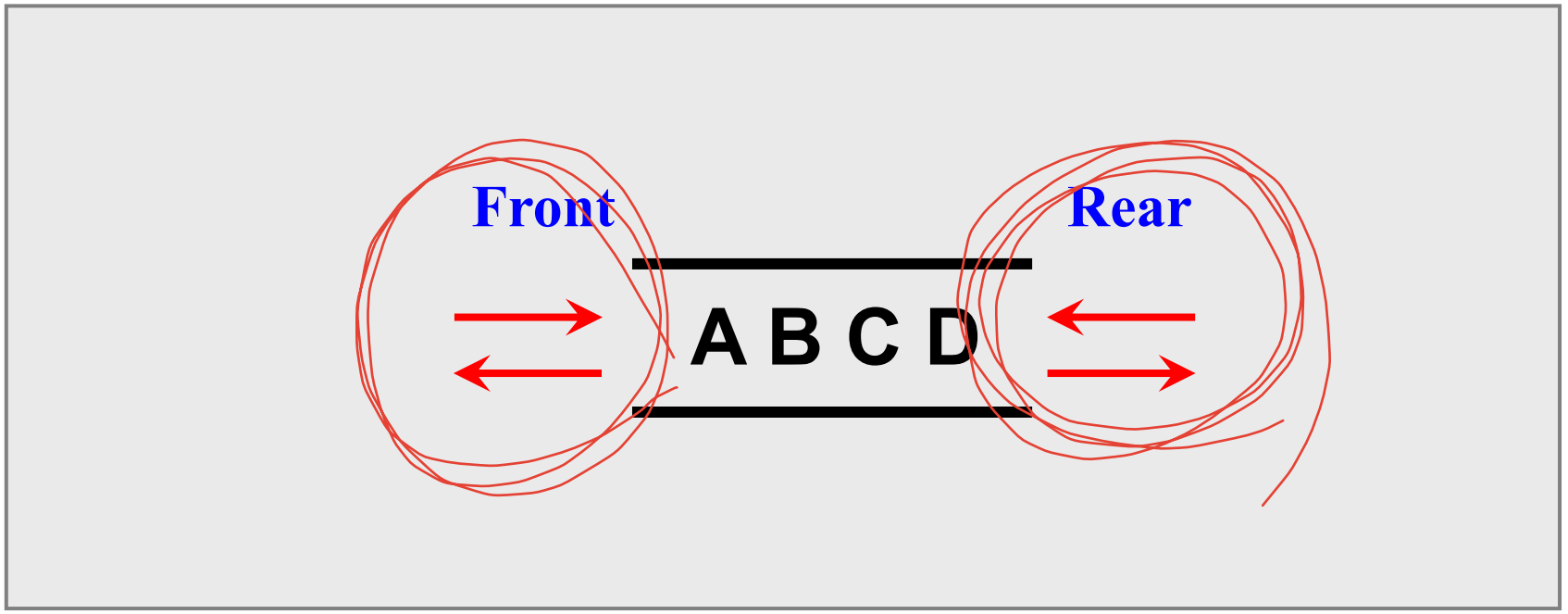
- **큐 구현:** 연결 자료구조

```
class Node :
    def __init__ (self, data, link=None):
        self.data = data
        self.link = link

class LinkedQueue :
    def __init__(self):
        self.__front = None
        self.__rear = None

    def enQueue(self, data) -> None:
    def deQueue(self):
    def peek(self):
    def isEmpty(self) -> bool:
    def printQueue(self):
```

Python

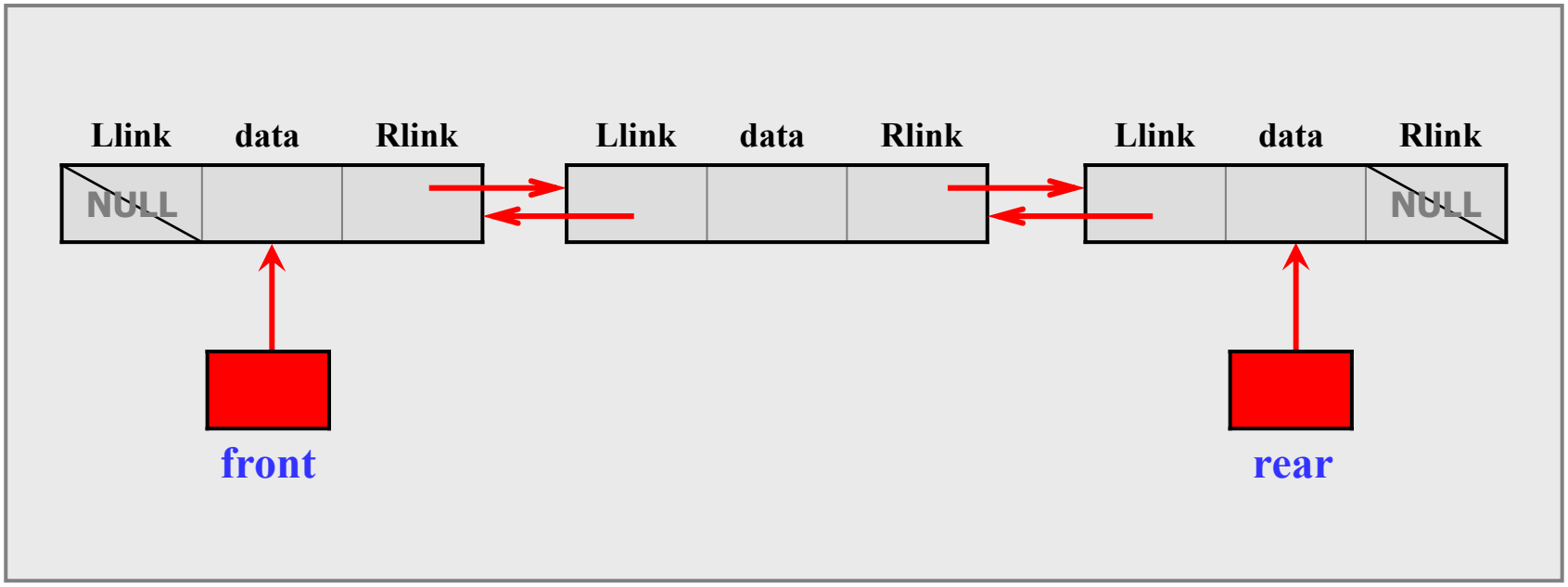
큐 구현: 연결 자료 구조 (5/6)

- **덱**(Deque, Double-ended Queue)
  - 큐의 양쪽 끝에서 삽입과 삭제가 모두 발생할 수 있는 큐
    - 스택과 큐의 성질을 모두 가지고 있는 자료구조



큐 구현: 연결 자료 구조 (6/6)

- 덱 구현
  - 이중 연결 리스트 구조를 이용한 덱의 구현



큐 응용

- 큐의 이해
- 큐의 구현
- **큐 응용**
  - 다양한 큐의 응용

큐 응용

다양한 큐의 응용

회문(回文, Palindromes)

- 앞에서 읽거나 뒤에서 읽으나 똑같은 단어나 문장
- 예: 똑바로 읽어도 거꾸로 읽어도 '기러기', '토마토', '우영우' 등

운영체제의 작업 큐

- 프로세스 스케줄링 큐(Process Scheduling Queue)
- 프린터 버퍼 큐(Printer Buffer Queue)

시뮬레이션에서의 큐잉 시스템

- 모의실험
- 큐잉 이론(Queueing Theory)
- 이벤트 발생시기
  - 시간 구동 시뮬레이션(Time-Driven Simulation)
  - 사건구동 시뮬레이션(Event-Driven Simulation)

참고문헌

[1] Michael T. Goodrich 외 2인 지음, 김유성 외 2인 옮김, “C++로 구현하는 자료구조와 알고리즘”, 한티에듀, 2020.

[2] “프로그래밍 대회 공략을 위한 알고리즘과 자료 구조 입문”, 와타노베 유타카 저, 윤인성 역, 인사이트, 2021.

[3] “IT CookBook, 쉽게 배우는 자료구조 with 파이썬”, 문병로, 한빛아카데미, 2022.

[4] “이것이 취업을 위한 코딩 테스트다 with 파이썬”, 나동빈, 한빛미디어, 2020.

[5] 문병로, “IT CookBook, 쉽게 배우는 알고리즘: 관계 중심의 사고법”(개정판), 개정판, 한빛아카데미, 2018.

[6] Richard E. Neapolitan, 도경구 역, “알고리즘 기초”, 도서출판 홍릉, 2017.

[7] 주우석, “IT CookBook, C・C++ 로 배우는 자료구조론”, 한빛아카데미, 2019.

[8] 이지영, “C 로 배우는 쉬운 자료구조”, 한빛아카데미, 2022.