

# 자료구조 1

최백준 [choi@startlink.io](mailto:choi@startlink.io)

---

# 스택

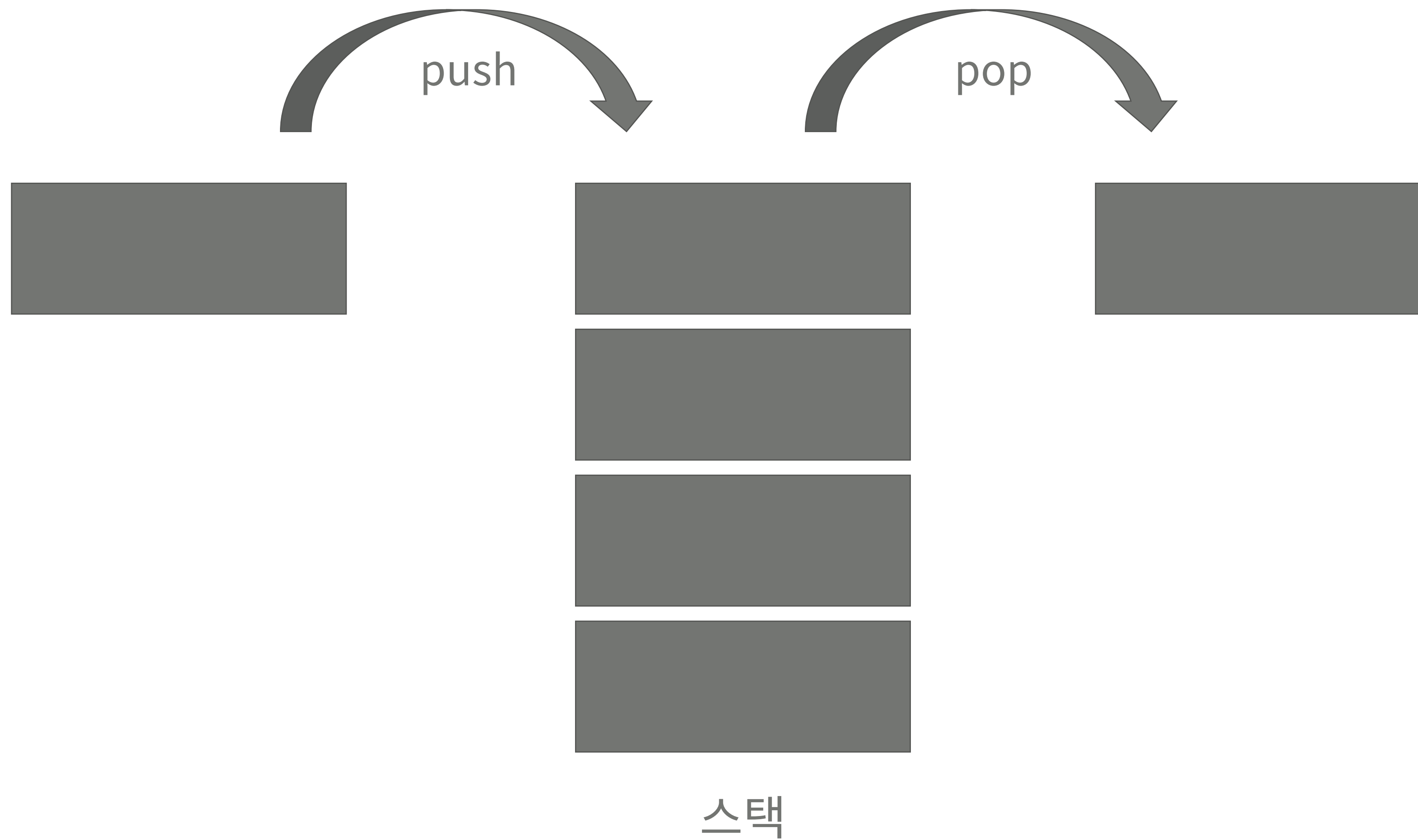
---

# 스택

Stack

3

- 스택은 한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조이다



# 스택

## Stack

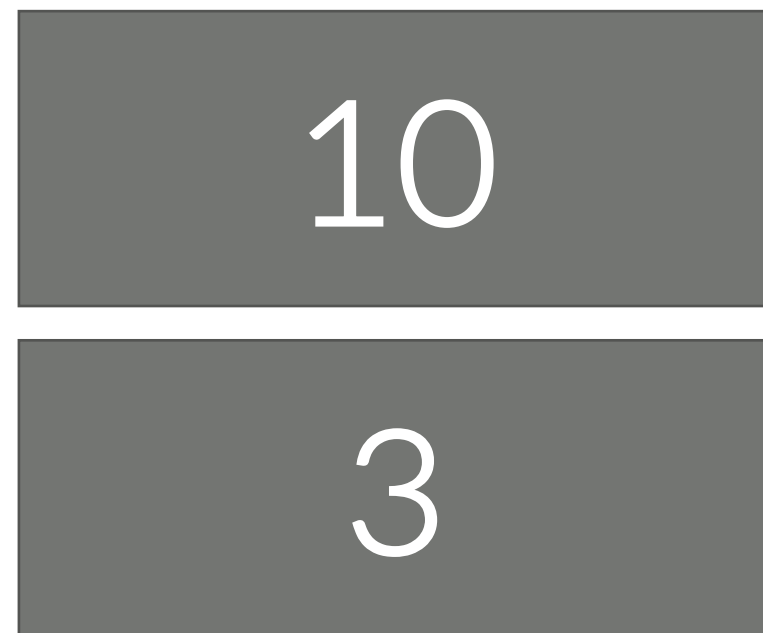
- 한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조
- 마지막으로 넣은 것이 가장 먼저 나오기 때문에 Last In Frist Out (LIFO) 라고도 한다.
- push: 스택에 자료를 넣는 연산
- pop: 스택에서 자료를 빼는 연산
- top: 스택의 가장 위에 있는 자료를 보는 연산
- empty: 스택이 비어있는지 아닌지를 알아보는 연산
- size: 스택에 저장되어있는 자료의 개수를 알아보는 연산

# 스택

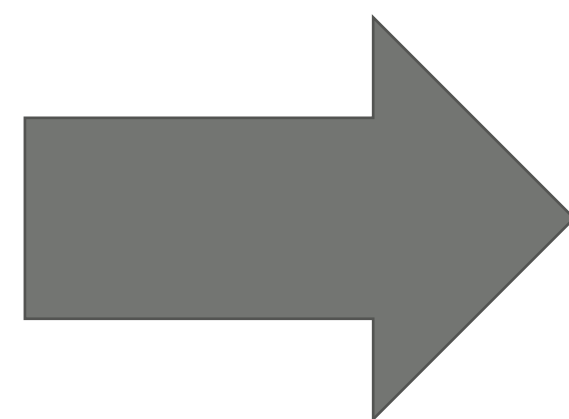
Stack

5

- 스택은 한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조이다



스택



push 5



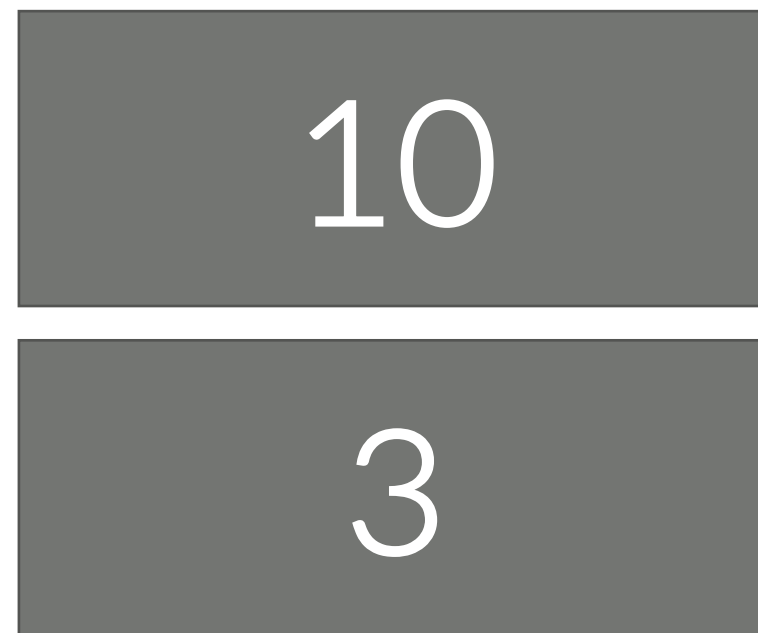
스택

# 스택

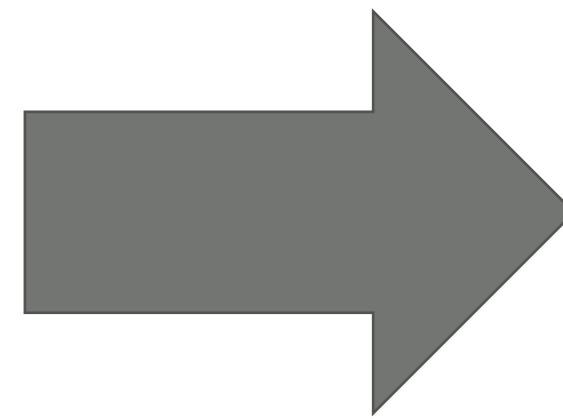
Stack

6

- 스택은 한쪽 끝에서만 자료를 넣고 뺄 수 있는 자료구조이다



스택



pop



스택

# 스택의 구현

## Stack

- 일차원 배열 하나로 구현할 수 있다.

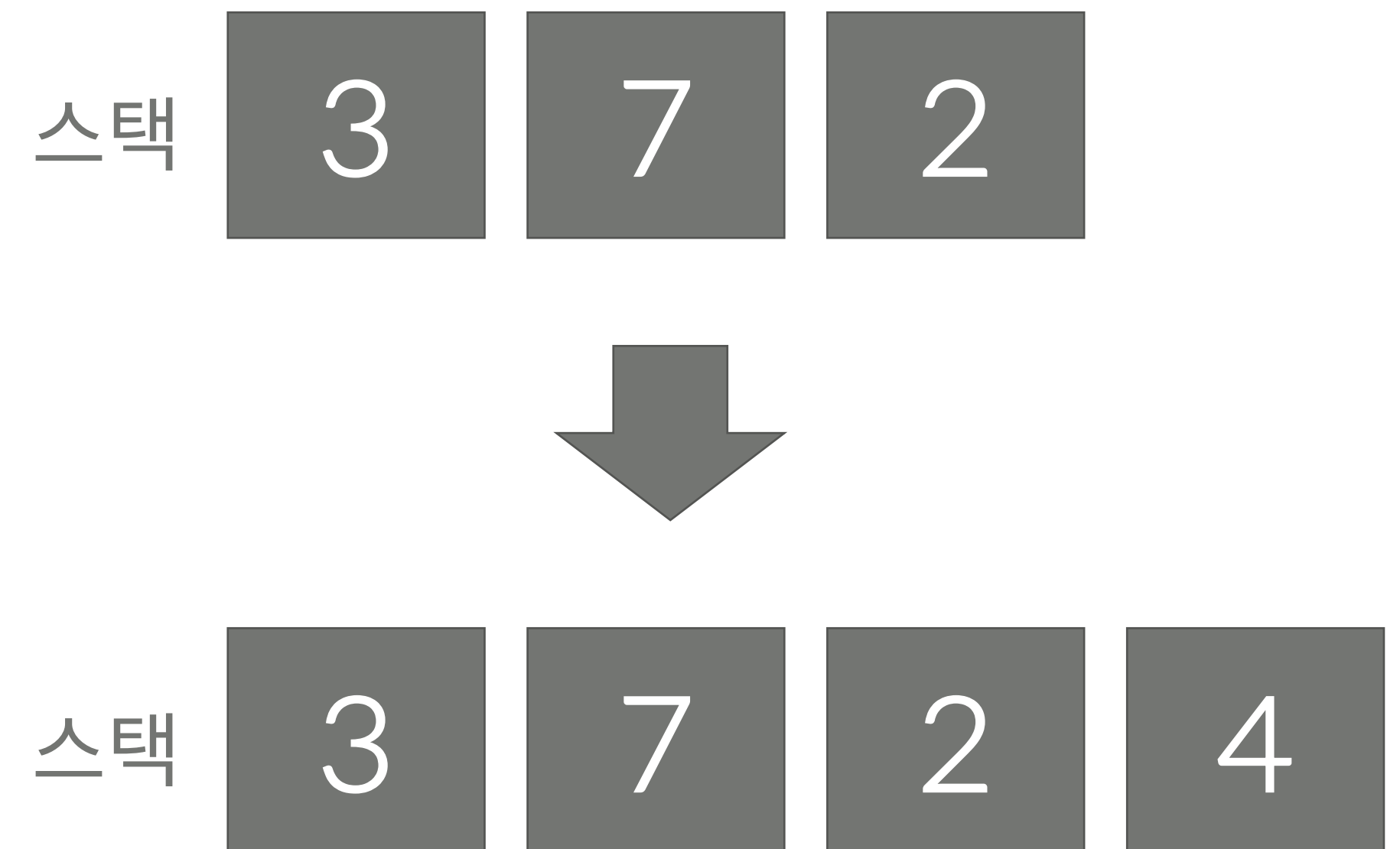
```
int stack[10000];
```

```
int size = 0;
```

# 스택의 구현

Stack

```
void push(int data) {  
    stack[size] = data;  
    size += 1;  
}
```

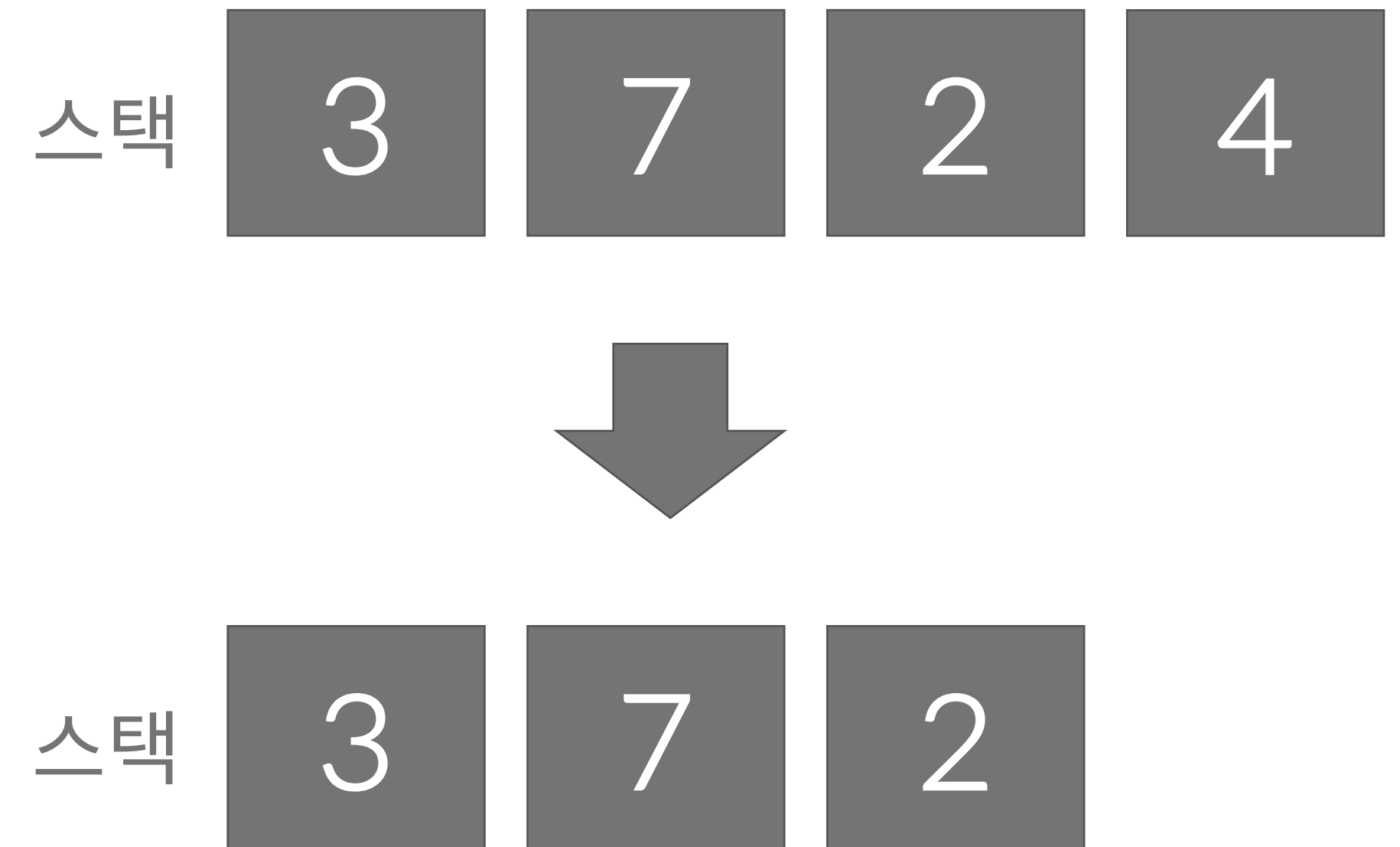




# 스택의 구현

Stack

```
int pop() {  
    stack[size-1] = 0;  
    size -= 1;  
}
```



# 스택

<https://www.acmicpc.net/problem/10828>

- 스택을 구현하는 문제
- 구현 소스: <http://codeplus.codes/905a5953c84340d1a573ba3466b16f83>
- 라이브러리 사용 소스: <http://codeplus.codes/cab3aefa6412478f8d81c7f37cc31721>

# 단어 뒤집기

<https://www.acmicpc.net/problem/9093>

- 문장이 주어졌을 때 단어를 모두 뒤집는 문제
- 단어의 순서를 바꿀 수 없고, 단어는 영어 알파벳으로만 이루어져 있다.
- 단어의 최대 길이: 20, 문장의 최대 길이: 1000

# 단어 뒤집기

<https://www.acmicpc.net/problem/9093>

- N개의 문자를 스택에 넣었다가 빼면 순서가 역순이 된다.
- 알파벳을 스택에 넣고, 공백이나 문자열의 끝이면 스택에서 모두 빼내서 역순을 만든다.

# 단어 뒤집기

<https://www.acmicpc.net/problem/9093>

- 소스: <http://codeplus.codes/dfcd4f2b762e4092862c8d6207eedaa5>

# 괄호

<https://www.acmicpc.net/problem/9012>

- 괄호 문자열이 주어졌을 때, 올바른 괄호 문자열인지 아닌지를 알아보는 문제
- 괄호 문자열: (와 )로만 이루어진 문자열
- 올바른 괄호 문자열: 괄호의 쌍이 올바른 문제

# 괄호

<https://www.acmicpc.net/problem/9012>

- 올바른 괄호 문자열의 예시
  - `()`
  - `(())()`
  - `((()))`
- 올바른 괄호 문자열이 아닌 예
  - `((()(`
  - `((())(())`
  - `((()`

# 괄호

<https://www.acmicpc.net/problem/9012>

- 스택을 사용해서 올바른 괄호 문자열인지 아닌지를 알 수 있다.
- (가 나오면 스택에 (를 넣고
- )가 나오면 스택에서 하나를 빼서 (인지 확인한다.
- 또는 하나를 뺄 수 있는지를 확인한다.



# 괄호

<https://www.acmicpc.net/problem/9012>

- `(( )) ( )`
- 스택:

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( ) ) ( )
- 스택: (
- ( ( ) ) ( )
- 스택: ( (
- ( ( ) ) ( )
- 스택: (
- ( ( ) ) ( )
- 스택:

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( ) ) ( ) )
- 스택: (
- ( ( ) ) ( ) )
- 스택:
- ( ( ) ) ( ) )
- 스택: ???
- 스택이 비어있는데 )가 나타났으므로 올바른 괄호 문자열이 아니다

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( ) ( ) ) ( ( ( ) ) )
- 스택:
- ( ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (
- ( ( ( ) ( ) ) ( ( ( ) ) )
- 스택: ((
- ( ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: ((
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택:
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: ((
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (((
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: ((
- ( ( ) ( ) ) ( ( ( ) ) )
- 스택: (

# 괄호

<https://www.acmicpc.net/problem/9012>

- `((()())((())))`
- 스택:
- 모든 과정이 끝났고, 스택이 비어있다!
- 올바른 괄호 문자열

# 괄호

<https://www.acmicpc.net/problem/9012>

- ( ( )
- 스택:
- ( ( )
- 스택: (
- ( ( )
- 스택: ((
- ( ( )
- 스택: (
- 끝났는데, 스택이 비어있지 않기 때문에 올바른 괄호 문자열이 아니다



# 괄호

<https://www.acmicpc.net/problem/9012>

- 과정을 살펴보면 스택에 들어가는 것이 여는 괄호 '(' 밖에 없다.
- 스택을 구현하지 않고, 스택의 크기만 이용해서, 몇 개의 여는 괄호가 스택에 들어가있는지를 판단하는 구현 방식도 가능하다.

# 괄호

<https://www.acmicpc.net/problem/9012>

- 소스: <http://codeplus.codes/668a325c1e064fffb1d2b74f076fadc6>

# 스택 수열

<https://www.acmicpc.net/problem/1874>

- 1부터 N까지의 수를 스택에 넣었다가 뽑아 놓음으로 하나의 수열을 만들 수 있다.
- push하는 순서는 오름차순이다.
- 임의의 수열 A가 있을 때, 스택을 이용해 이 수열을 만들 수 있는지 없는지 구하는 문제
- $A = [4, 3, 6, 8, 7, 5, 2, 1]$  이라면
- `++++--++-+-`를 이용해서 만들 수 있다.

# 스택 수열

<https://www.acmicpc.net/problem/1874>

- 스택에 push되는 순서는 오름차순이다.
- pop되는 순서대로 수열 A가 만들어지기 때문에, A의 첫 수부터 순서대로 만들어보면 된다.
- $m$  = 스택에 추가되어야 하는 수
- $A[i]$  = 만들어야 하는 수열 A의 수

# 스택 수열

<https://www.acmicpc.net/problem/1874>

- $m =$  스택에 추가되어야 하는 수
- $A[i] =$  만들어야 하는 수열  $A$ 의 수
- $m < A[i]$ :  $A[i]$ 를 pop해야 하기 때문에,  $m$ 부터  $A[i]$ 까지의 모든 수를 순서대로 push해야 한다.  
이후 상태는  $m == A[i]$ 인 경우와 같다.
- $m > A[i]$ 인 경우: 불가능한 경우다. pop을 하면  $A[i]$ 가 아닌 다른 수가  $A$ 에 추가된다.
- $m == A[i]$ 인 경우: pop을 해서  $A[i]$ 를 만들면 된다.

# 스택 수열

<https://www.acmicpc.net/problem/1874>

- 소스: <http://codeplus.codes/4799b36eb5dd488ea4c291751e381da8>

# 에디터

<https://www.acmicpc.net/problem/1406>

- 커서: 문장의 맨 앞, 맨 뒤, 문장 중간 임의의 곳에 위치할 수 있다.
- L: 커서를 왼쪽으로 한 칸 옮김
- D: 커서를 오른쪽으로 한 칸 옮김
- B: 커서 왼쪽에 있는 문자를 삭제함
- P \$: \$라는 문자를 커서 오른쪽에 추가함. 커서는 \$에 위치함

# 에디터

<https://www.acmicpc.net/problem/1406>

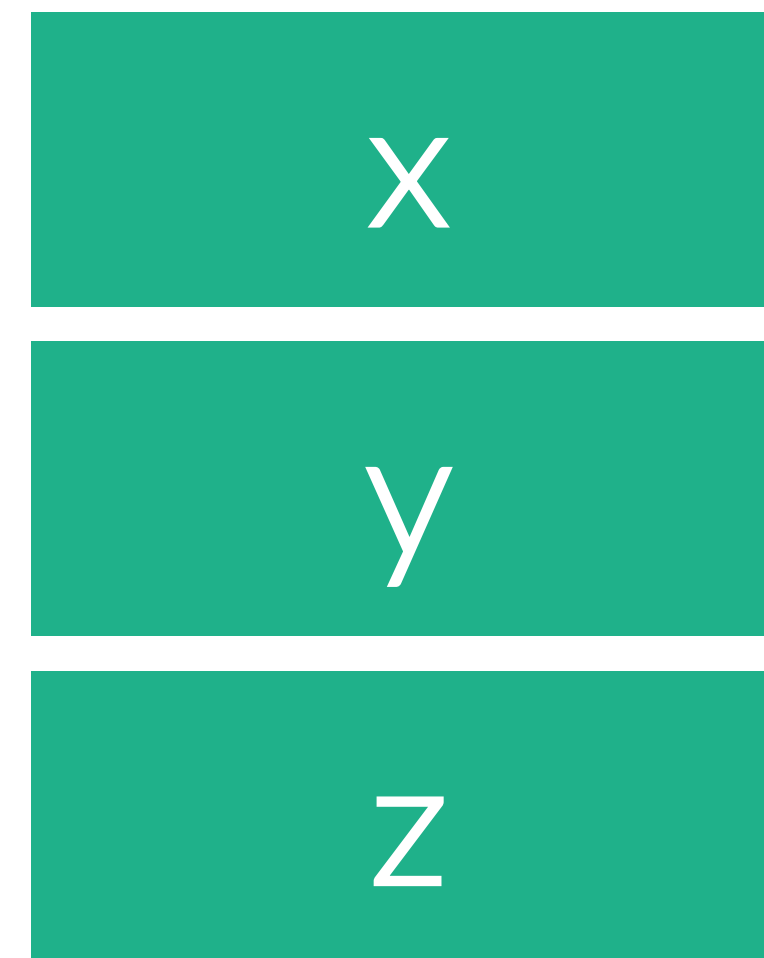
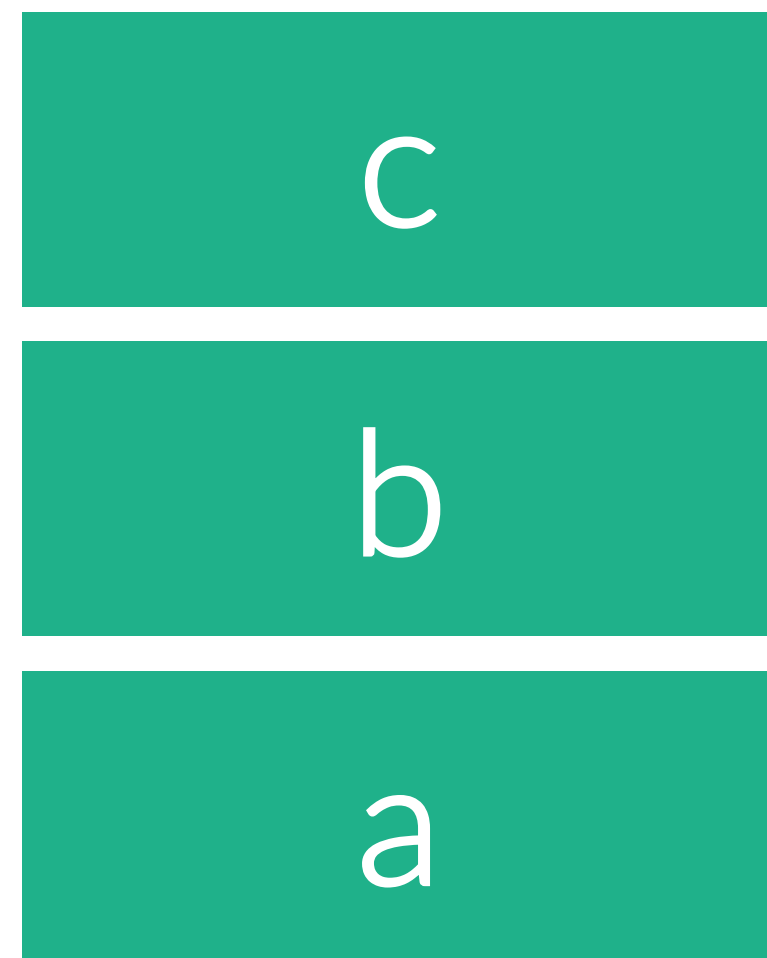
- 문자열의 길이를 M이라고 하자.
- L 연산:  $abc|xyz \rightarrow ab|cxyz$
- D 연산:  $ab|cxyz \rightarrow abc|xyz$
- B 연산:  $abc|xyz \rightarrow ab|xyz$
- P d 연산:  $ab|xyz \rightarrow abd|xyz$
- 이를 문자열을 이용해 구현하면 연산 하나의 시간복잡도는  $O(M)$ 이 된다.
- M은 최대 60만까지 될 수 있기 때문에, 시간이 너무 오래 걸린다.



# 에디터

<https://www.acmicpc.net/problem/1406>

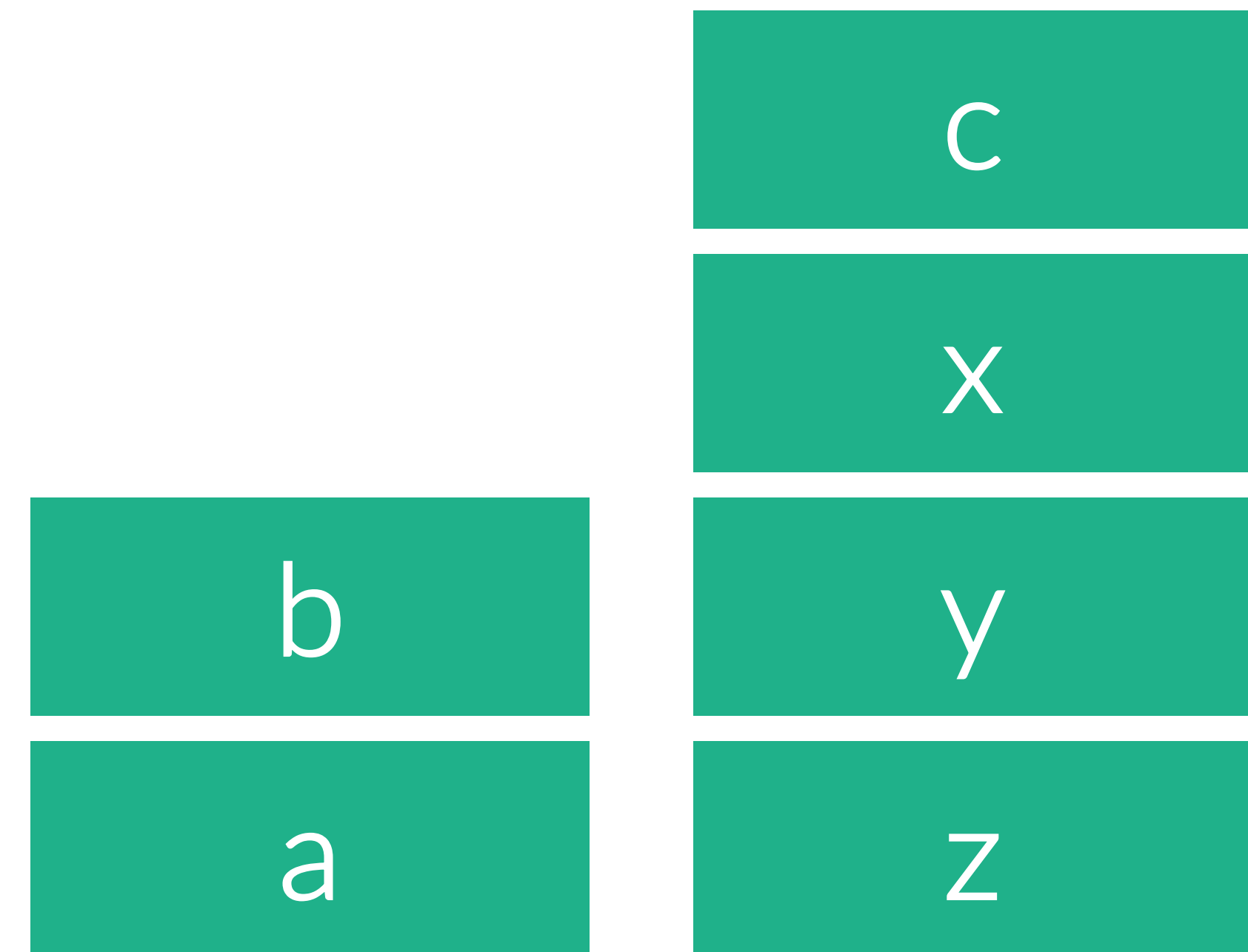
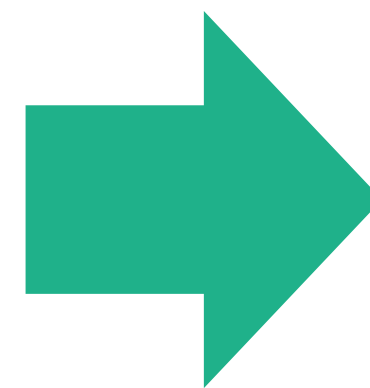
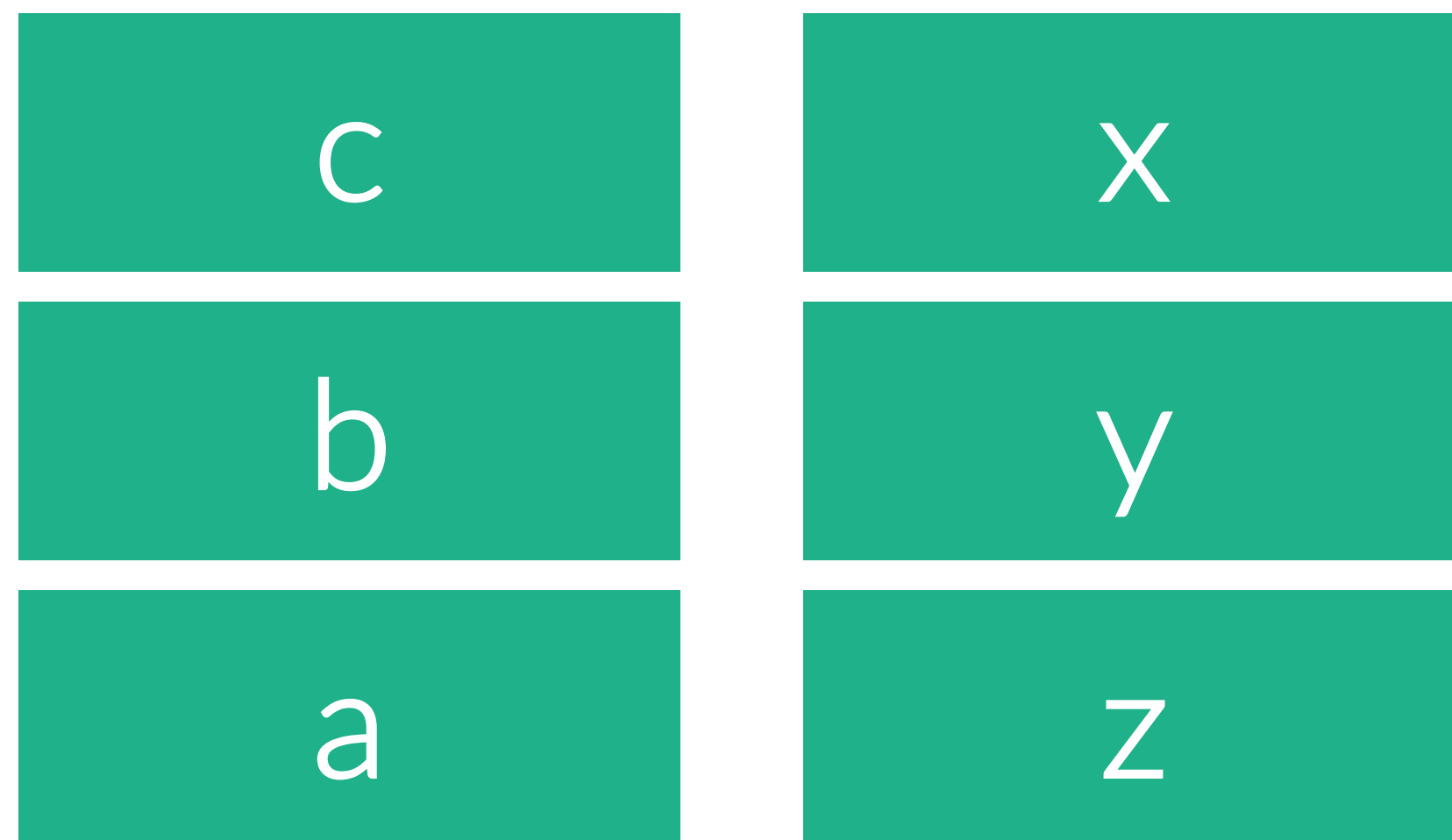
- 커서를 기준으로 커서의 왼쪽 스택(left)와 오른쪽 스택(right)로 나눠서 문제를 풀 수 있다.
- 예: abc | xyz (|는 커서)



# 에디터

<https://www.acmicpc.net/problem/1406>

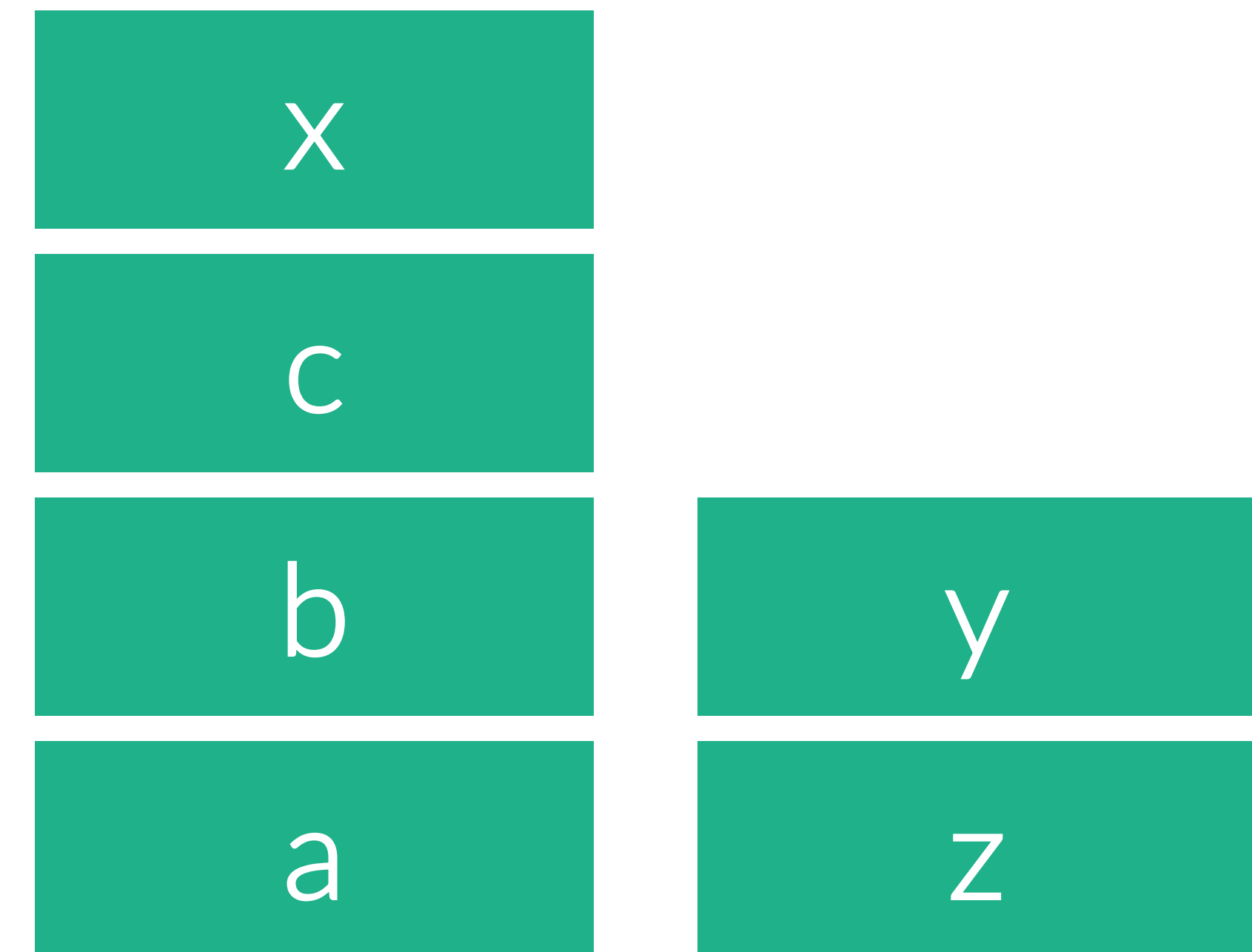
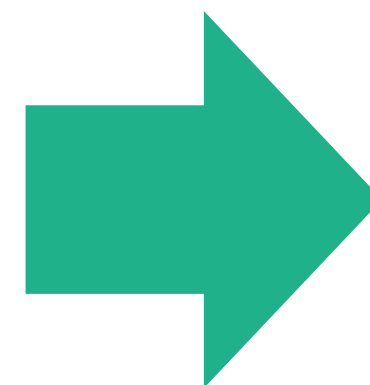
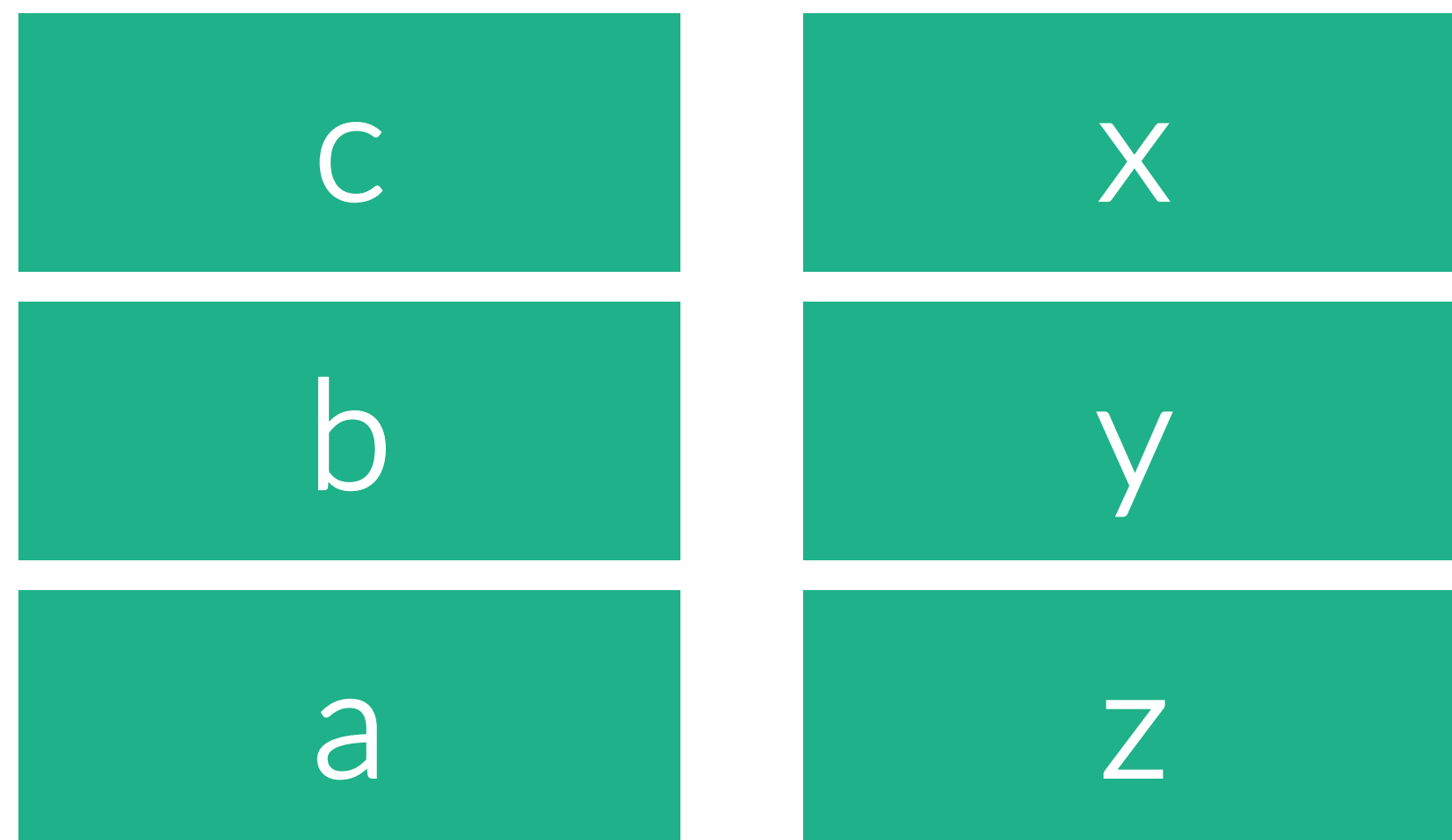
- L: 커서를 왼쪽으로 한 칸 옮김
- `abc | xyz -> ab | cxyz`



# 에디터

<https://www.acmicpc.net/problem/1406>

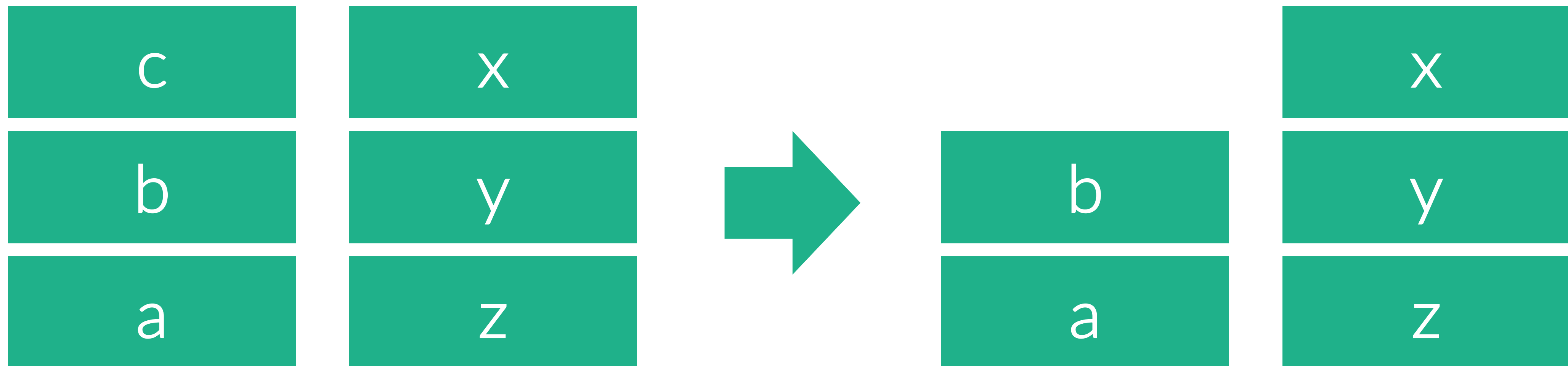
- D: 커서를 오른쪽으로 한 칸 옮김
- `abc | xyz -> abcx | yz`



# 에디터

<https://www.acmicpc.net/problem/1406>

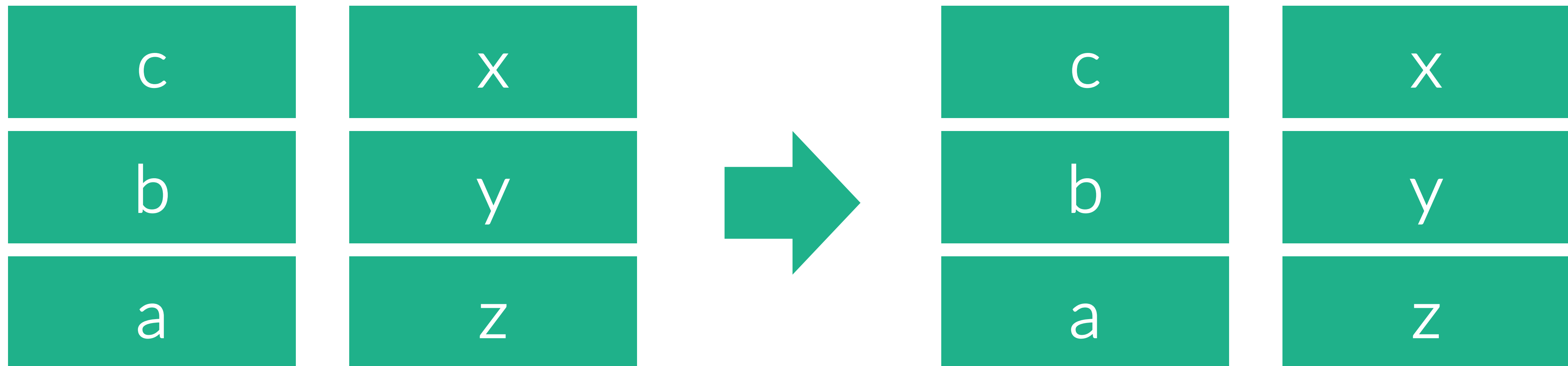
- B: 커서를 왼쪽에 있는 문자를 삭제함
- $abc \mid xyz \rightarrow ab \mid xyz$



# 에디터

<https://www.acmicpc.net/problem/1406>

- P \$: \$를 커서 오른쪽에 추가하고 커서는 \$의 오른쪽에 위치함
- abc | xyz -> abcd | xyz



# 에디터

<https://www.acmicpc.net/problem/1406>

- 이 문제는 링크드 리스트를 이용해서도 풀 수 있다.

# 에디터

<https://www.acmicpc.net/problem/1406>

- 소스: <http://codeplus.codes/3209fe229a524039a8813f51f5a59ad8>

큐

---



# 큐

## Queue

- 한쪽 끝에서만 자료를 넣고 다른 한쪽 끝에서만 뺄 수 있는 자료구조
- 먼저 넣은 것이 가장 먼저 나오기 때문에 First In Frist Out (FIFO) 라고도 한다.
- push: 큐에 자료를 넣는 연산
- pop: 큐에서 자료를 빼는 연산
- front: 큐의 가장 앞에 있는 자료를 보는 연산
- back: 큐의 가장 뒤에 있는 자료를 보는 연산
- empty: 큐가 비어있는지 아닌지를 알아보는 연산
- size: 큐에 저장되어있는 자료의 개수를 알아보는 연산

# 큐

## Queue

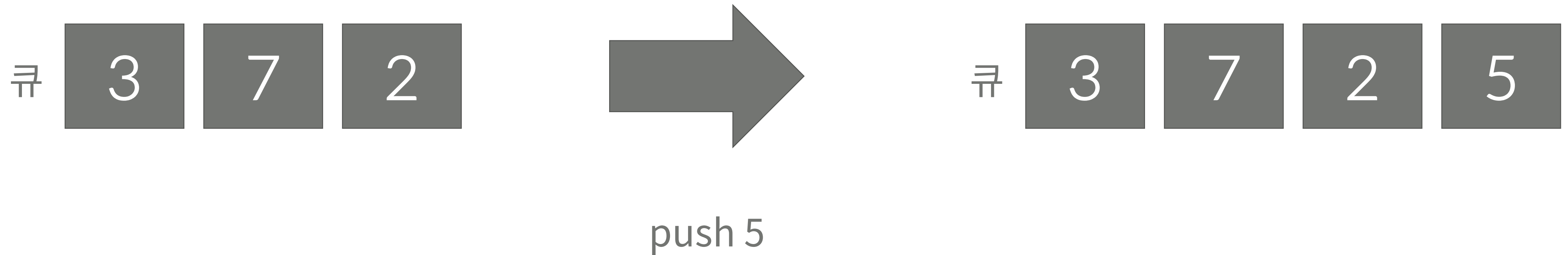
- 스택은 C++의 경우에는 STL의 queue
- Java의 경우에는 `java.util.Queue`을 사용하는 것이 좋다.

# 큐

Queue

43

- 한쪽 끝에서만 자료를 넣고 다른 한쪽 끝에서만 뺄 수 있는 자료구조

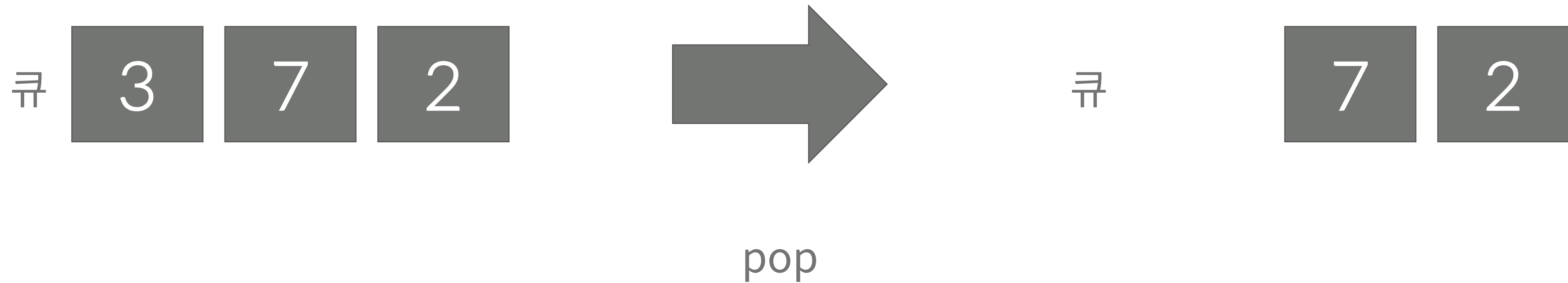


# 큐

Queue

44

- 한쪽 끝에서만 자료를 넣고 다른 한쪽 끝에서만 뺄 수 있는 자료구조



# 큐의 구현

## Queue

- 일차원 배열 하나로 구현할 수 있다. (큐에 포함되어있는 내용은 [begin,end) 이다)

```
int queue[10000];
```

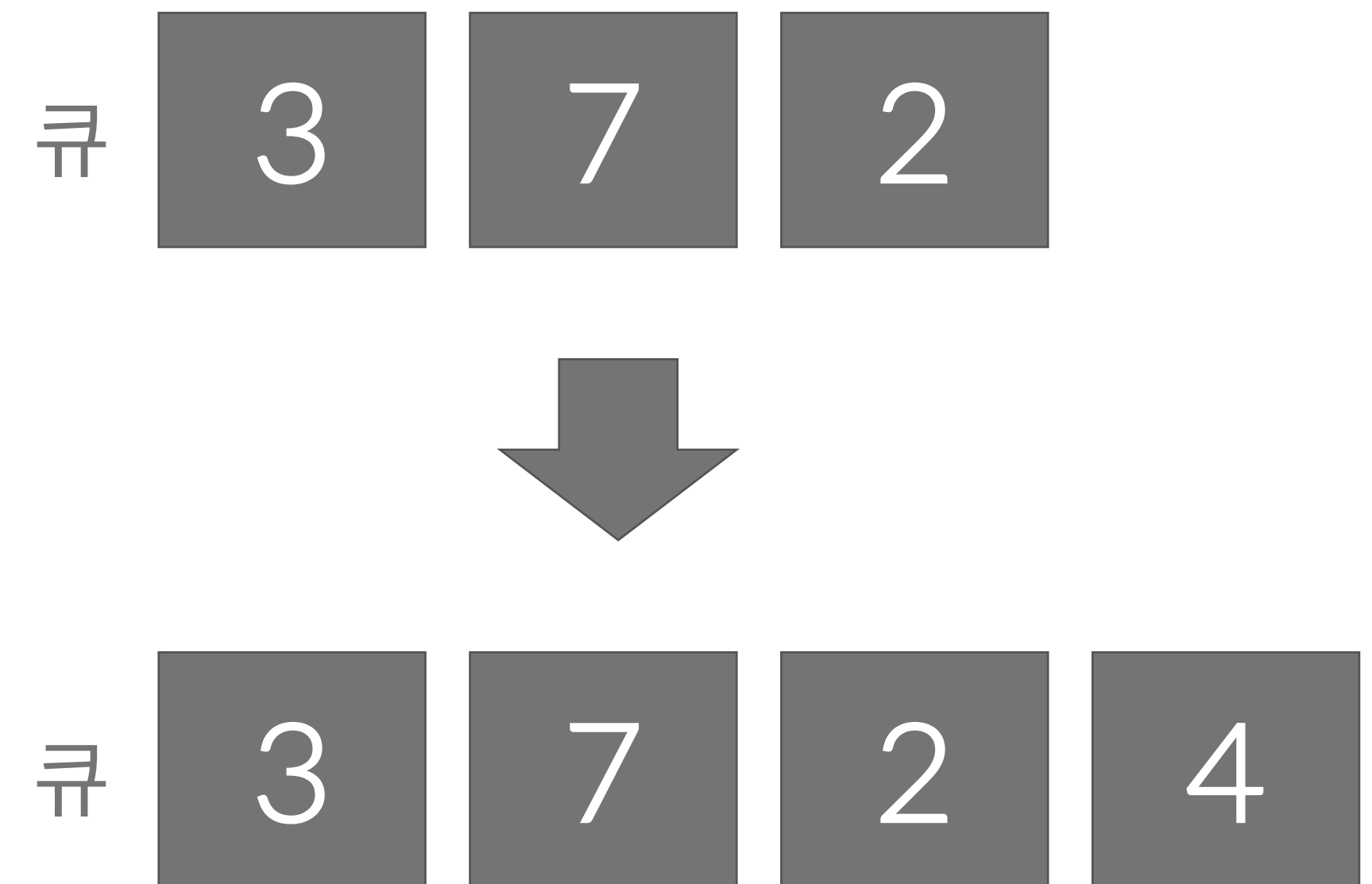
```
int begin = 0;
```

```
int end = 0;
```

# 큐의 구현

Queue

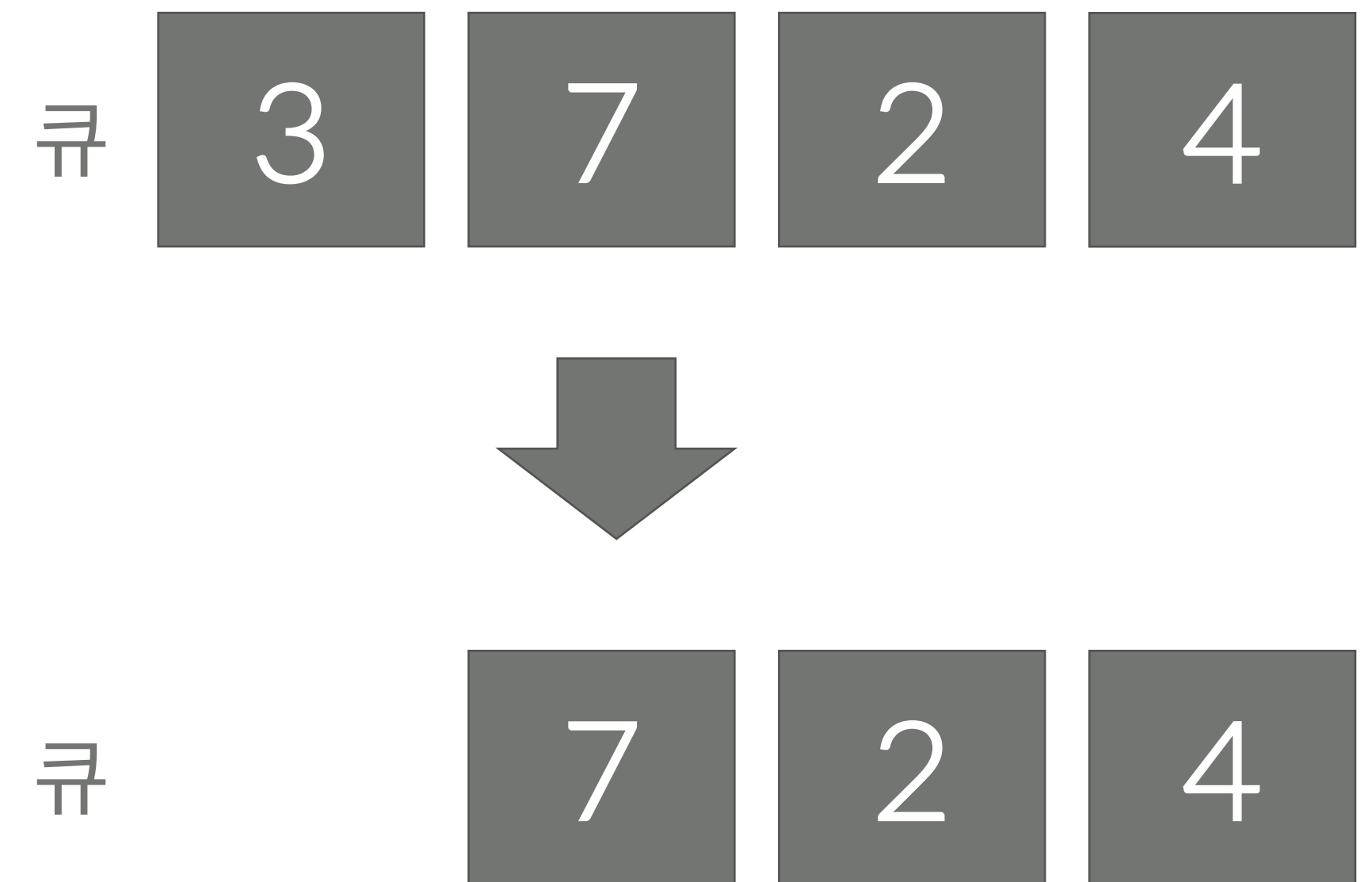
```
void push(int data) {  
    queue[end] = data;  
    end += 1;  
}
```



# 큐의 구현

Queue

```
int pop() {  
    queue[begin] = 0;  
    begin += 1;  
}
```



# 큐의 구현

Queue

```
bool empty() {  
    if (begin == end) return true;  
    else return false;  
}  
  
int size() {  
    return end-begin;  
}
```



# 큐

<https://www.acmicpc.net/problem/10845>

- 큐를 구현하는 문제

# 큐

<https://www.acmicpc.net/problem/10845>

- 큐를 구현하는 문제
- 구현 소스: <http://codeplus.codes/32a037ff0b84435bb54a88937003110d>
- 라이브러리 소스: <http://codeplus.codes/d4d22d8cfcb94e54b68b6bb4b2910437>

# 조세퍼스 문제

<https://www.acmicpc.net/problem/1158>

- 1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수  $M(\leq N)$ 이 주어진다
- 이제 순서대로 M번째 사람을 제거한다
- 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다
- 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다
- 원에서 사람들이 제거되는 순서를 (N, M)-조세퍼스 순열이라고 한다

# 조세퍼스 문제

<https://www.acmicpc.net/problem/1158>

- 문제에 나와있는 대로 큐를 이용해서 시뮬레이션을 하면 된다
- $N = 7$
- $M = 3$

1	2	3	4	5	6	7
---	---	---	---	---	---	---

# 조세퍼스 문제

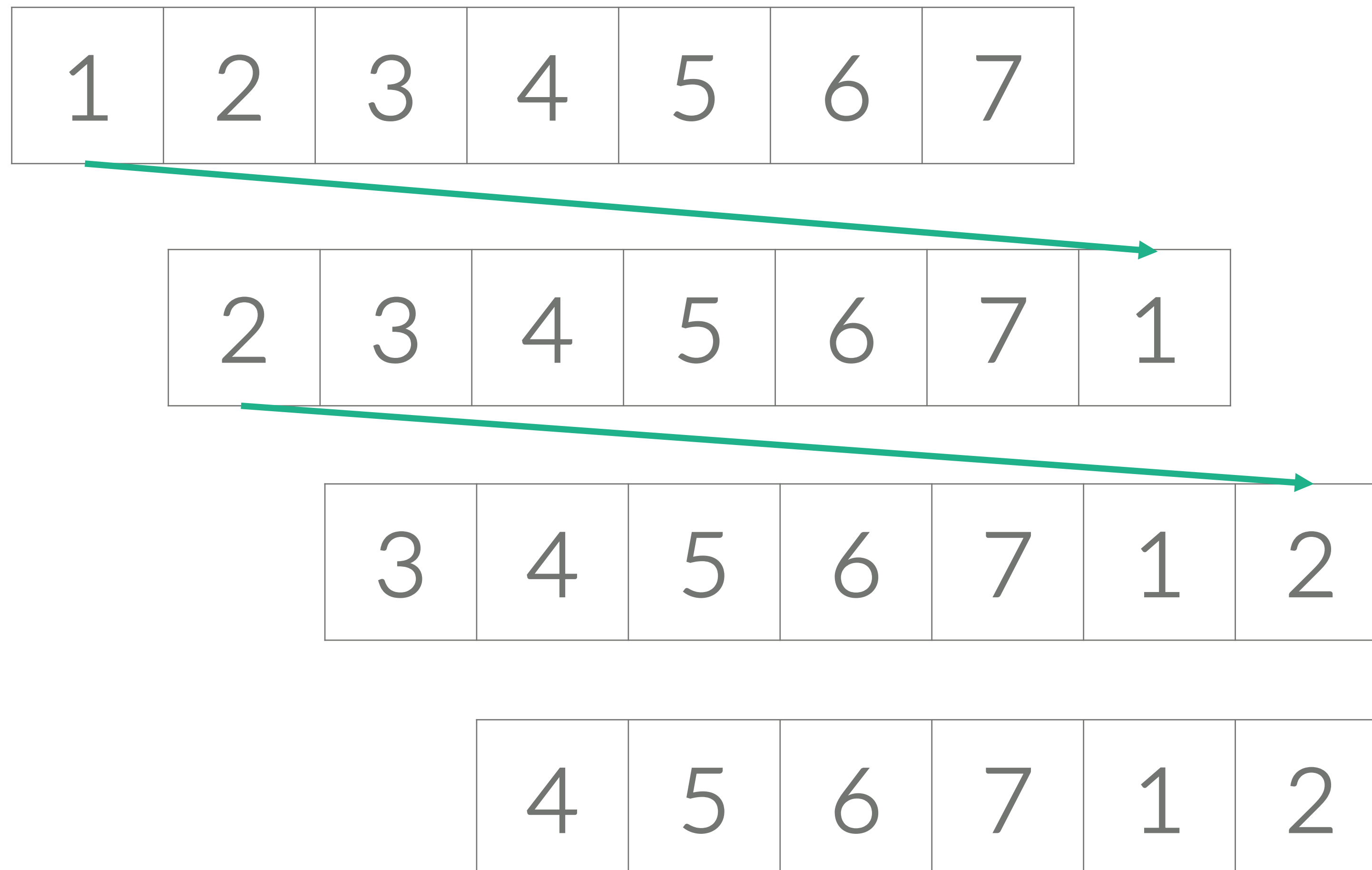
53

<https://www.acmicpc.net/problem/1158>

- 문제에 나와있는 대로 큐를 이용해서 시뮬레이션을 하면 된다

- $N = 7$

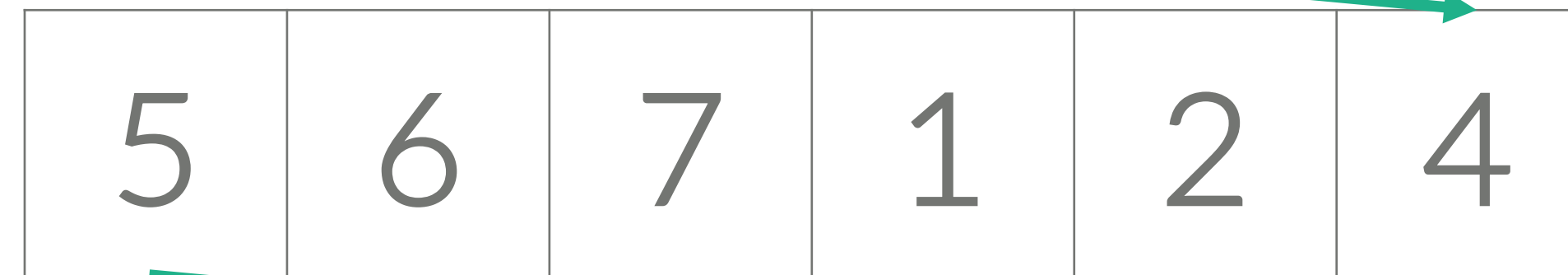
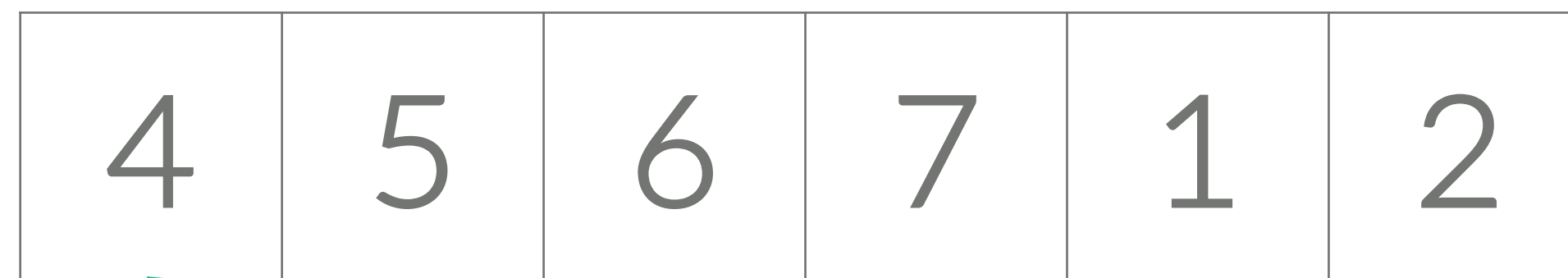
- $M = 3$



# 조세퍼스 문제

<https://www.acmicpc.net/problem/1158>

- 문제에 나와있는 대로 큐를 이용해서 시뮬레이션을 하면 된다
- $N = 7$
- $M = 3$



# 조세퍼스 문제

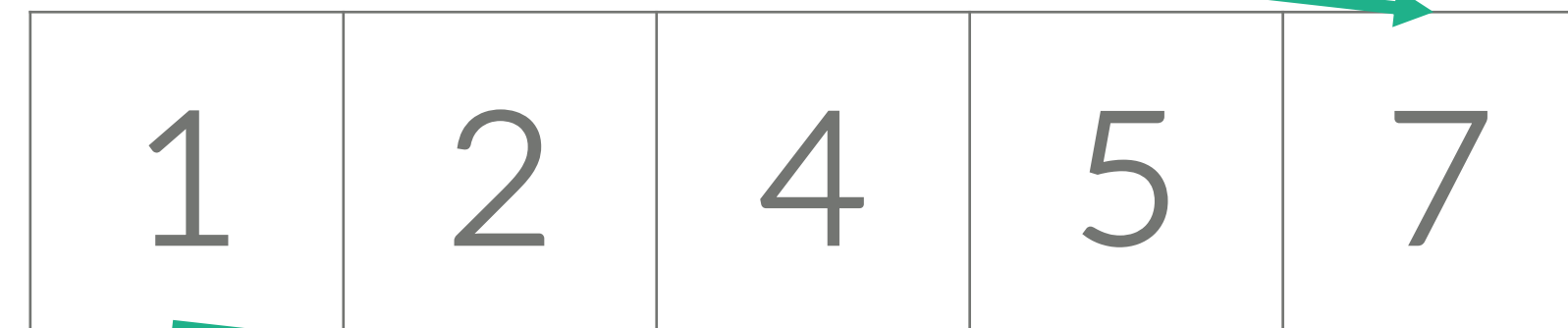
55

<https://www.acmicpc.net/problem/1158>

- 문제에 나와있는 대로 큐를 이용해서 시뮬레이션을 하면 된다

- $N = 7$

- $M = 3$



# 조세퍼스 문제

<https://www.acmicpc.net/problem/1158>

- 소스: <http://codeplus.codes/3c56720a0010405291c16011137b4e9c>



데

---

# 덱

## Deque

- 양 끝에서만 자료를 넣고 양 끝에서 뺄 수 있는 자료구조
- Double-ended queue의 약자이다.
- push\_front: 덱의 앞에 자료를 넣는 연산
- push\_back: 덱의 뒤에 자료를 넣는 연산
- pop\_front: 덱의 앞에서 자료를 빼는 연산
- pop\_back: 덱의 두에서 자료를 빼는 연산
- front: 덱의 가장 앞에 있는 자료를 보는 연산
- back: 덱의 가장 뒤에 있는 자료를 보는 연산

# 덱

## Deque

- 양 끝에서만 자료를 넣고 양 끝에서 뺄 수 있는 자료구조
- Double-ended queue의 약자이다.
- push\_front: 덱의 앞에 자료를 넣는 연산
- push\_back: 덱의 뒤에 자료를 넣는 연산
- pop\_front: 덱의 앞에서 자료를 빼는 연산
- pop\_back: 덱의 두에서 자료를 빼는 연산
- front: 덱의 가장 앞에 있는 자료를 보는 연산
- back: 덱의 가장 뒤에 있는 자료를 보는 연산

# 덱

<https://www.acmicpc.net/problem/10866>

- 덱을 구현하는 문제

<https://www.acmicpc.net/problem/10866>

- 소스: <http://codeplus.codes/05505b4020f64d32a06a15e813e8ebf7>

끝

---

# 코드 플러스

<https://code.plus>

- 슬라이드에 포함된 소스 코드를 보려면 "정보 수정 > 백준 온라인 저지 연동"을 통해 연동한 다음, "백준 온라인 저지"에 로그인해야 합니다.
- 강의 내용에 대한 질문은 코드 플러스의 "질문 게시판"에서 할 수 있습니다.
- 문제와 소스 코드는 슬라이드에 첨부된 링크를 통해서 볼 수 있으며, "백준 온라인 저지"에서 서비스됩니다.
- 슬라이드와 동영상 강의는 코드 플러스 사이트를 통해서만 볼 수 있으며, 동영상 강의의 녹화와 다운로드, 배포와 유통은 저작권법에 의해서 금지되어 있습니다.
- 다른 경로로 이 슬라이드나 동영상 강의를 본 경우에는 [codeplus@startlink.io](mailto:codeplus@startlink.io) 로 이메일 보내주세요.
- 강의 내용, 동영상 강의, 슬라이드, 첨부되어 있는 소스 코드의 저작권은 스타트링크와 최백준에게 있습니다.