

# 알튜비튜

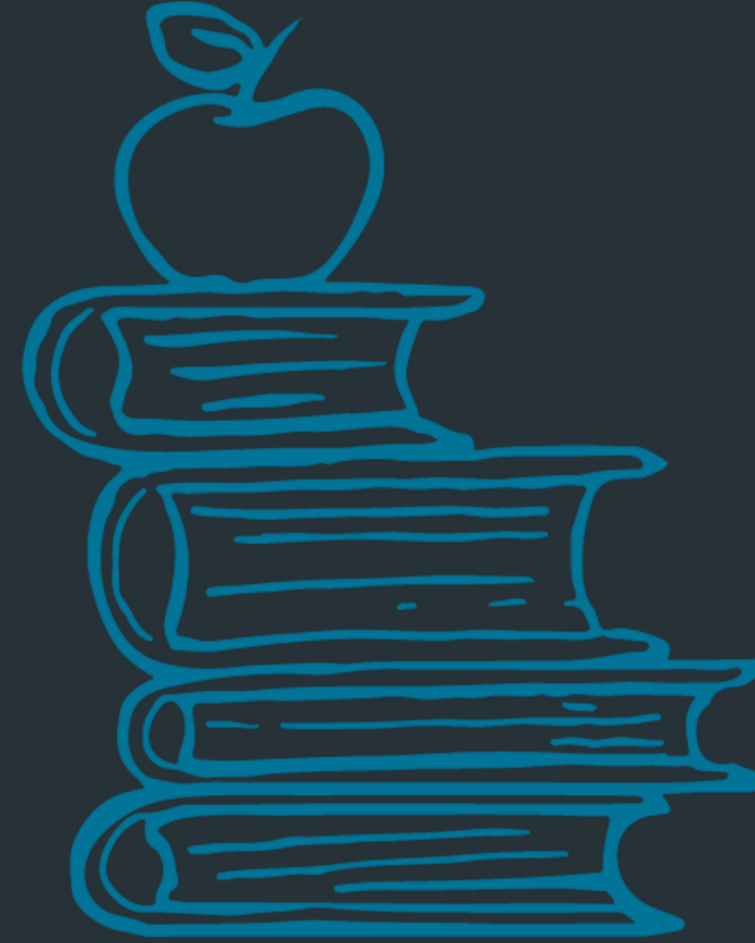
## 스택, 큐, 덱

오늘은 STL에서 제공하는 container adaptor인 stack과 queue 그리고 sequence container인 deque에 대해 알아봅니다.  
가장 대표적이면서 가장 중요하기도 한 자료구조들 입니다.

STL : 데이터를 저장하는 객체를 의미/ 데이터를 효과적으로 저장하기 위한 다양한 자료구조 구현체 제공  
Container adapter: 기존 컨테이너를 변경(modify, adapt)하여 특정 인터페이스(기능)만을 제공하도록 만든 컨테이너  
Sequence Container: 데이터를 선형 자료구조로 저장하는 컨테이너

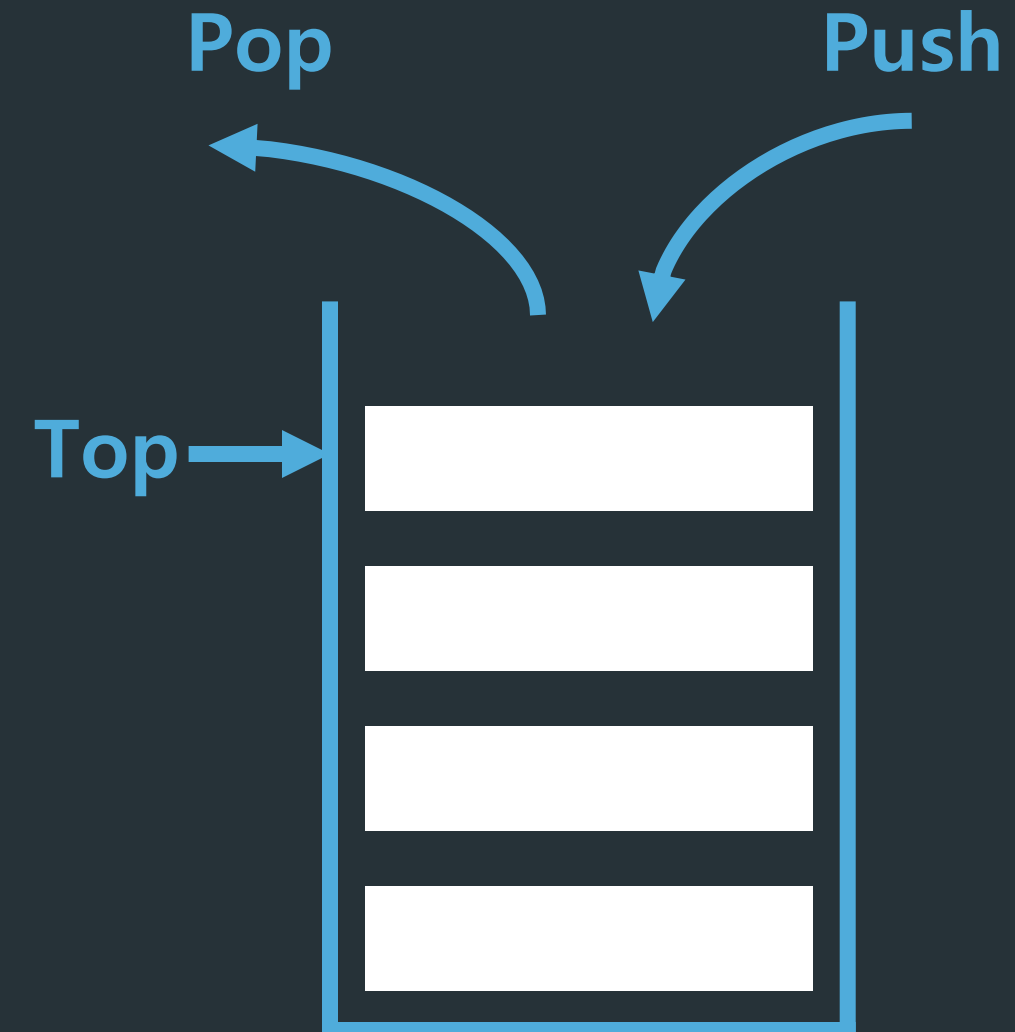
어디까지 먹어봤니?  
내가 모르는 프링글스의 세상!

**프링글스**

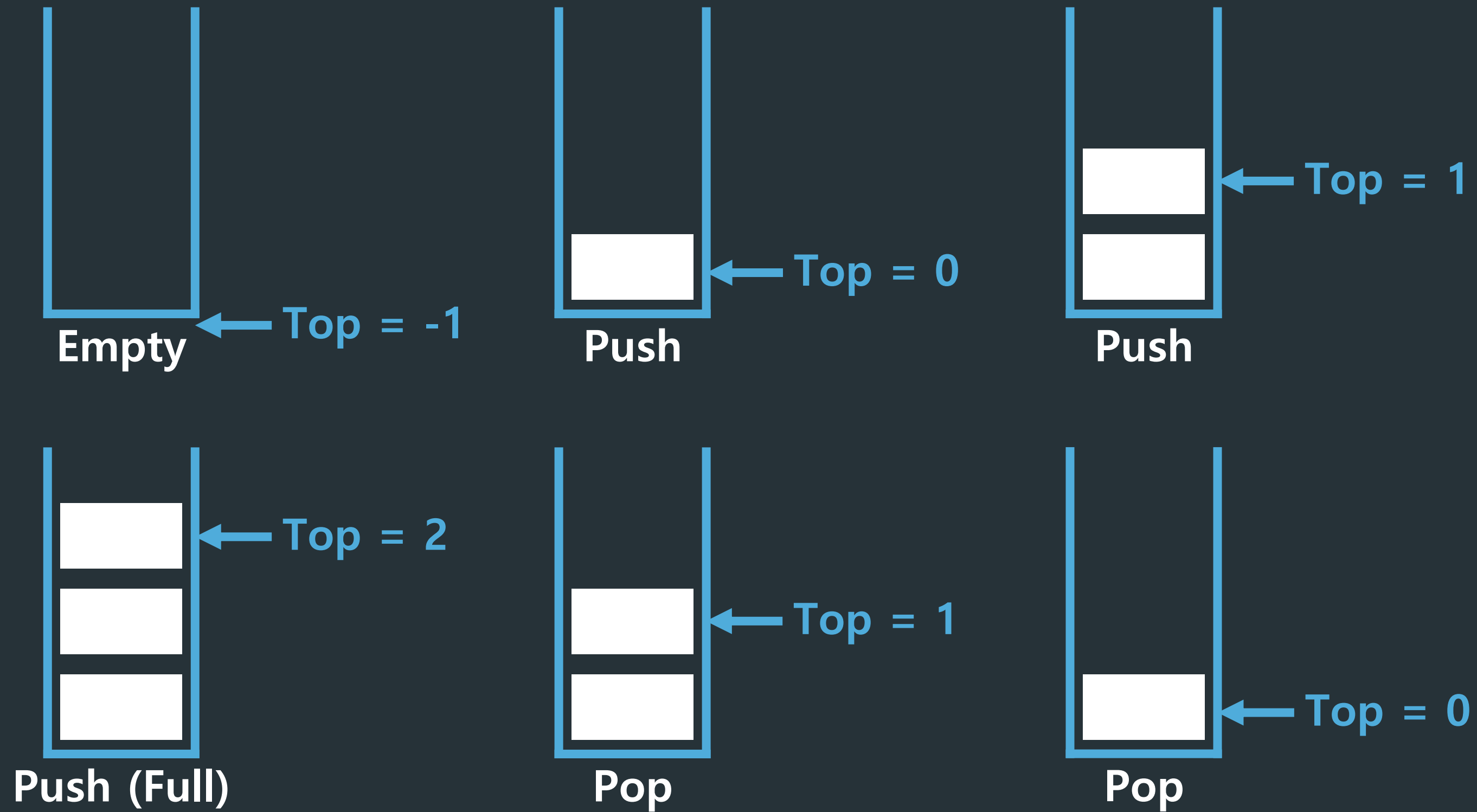


## Stack

- LIFO (Last In First Out)
- 자료의 맨 끝 위치에서만 모든 연산이 이루어짐
- 모든 연산에 대한 시간 복잡도는  $O(1)$
- 연산이 이루어지는 위치를 *top*이라고 부름
- 삽입은 *push*, 삭제는 *pop*



# 배열로 크기 3인 스택 구현하기



## /<> 10828번 : 스택 – Silver 4

### 문제

- 다음의 명령을 처리하는 스택 프로그램 만들기
  1. `push X` : 정수 X를 스택에 **삽입**
  2. `pop` : 스택에서 가장 위에 있는 정수를 **빼고**, 출력. 스택이 비었다면 -1 출력
  3. `size` : 스택에 들어있는 정수의 **개수** 출력
  4. `empty` : 스택이 **비었으면 1**, 아니라면 **0**을 출력
  5. `top` : 스택의 가장 위에 있는 정수를 **출력**. 스택이 비었다면 -1 출력

### 제한 사항

- 명령의 수 N의 범위는  $1 \leq N \leq 10,000$
- 명령과 함께 주어지는 정수 k의 범위는  $1 \leq k \leq 100,000$

## 예제 입력1

```
14
push 1
push 2
top
size
empty
pop
pop
pop
size
empty
pop
push 3
empty
top
```

## 예제 출력1

```
2
2
0
2
1
-1
0
1
-1
0
3
```



# 스택을 쓸 일이 얼마나 많은데...!



Search:  Go

Not logged in

register log in

Reference <stack> stack

You were redirected to [cplusplus.com/stack](https://cplusplus.com/stack) || See search results for: "stack"

C++

Information

Tutorials

Reference

Articles

Forum

Reference

C library:

Containers:

<array>

<deque>

<forward\_list>

<list>

<map>

<queue>

<set>

<stack>

<unordered\_map>

<unordered\_set>

<vector>

Input/Output:

Multi-threading:

Other:

<stack>

stack

stack

stack::stack

member functions:

stack::emplace

stack::empty

stack::pop

stack::push

stack::size

stack::swap

class template

std::stack

&lt;stack&gt;

template &lt;class T, class Container = deque&lt;T&gt; &gt; class stack;

LIFO stack

Stacks are a type of container adaptor, specifically designed to operate in a LIFO context (last-in first-out), where elements are inserted and extracted only from one end of the container.

stacks are implemented as *container adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *pushed/popped* from the "back" of the specific container, which is known as the *top* of the stack.

The underlying container may be any of the standard container class templates or some other specifically designed container class. The container shall support the following operations:

- empty
- size
- back
- push\_back
- pop\_back

The standard container classes `vector`, `deque` and `list` fulfill these requirements. By default, if no container class is specified for a particular stack class instantiation, the standard container `deque` is used.

Template parameters

T

Type of the elements.  
Aliased as member type `stack::value_type`.

Container

Type of the internal *underlying container* object where the elements are stored.  
Its `value_type` shall be T.  
Aliased as member type `stack::container_type`.

## std::stack

- push(element): top에 원소를 추가
- pop(): top에 있는 원소를 삭제
- top(): top에 있는 원소를 반환
- empty(): 스택이 비어있는지 확인 (비어있으면 true)
- size(): 스택 사이즈를 반환

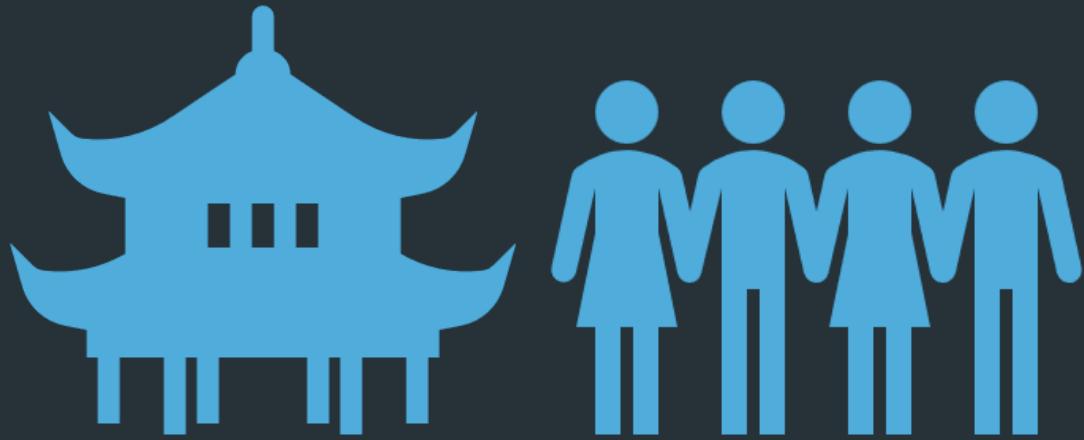


# 파이썬은요...?

- 파이썬은 따로 Stack 자료구조를 제공하지 X
- 이미 list에 모두 구현되어 있음

## list()

- stack = list() 일 때...
- stack.append(element): top에 원소를 추가
- stack.pop(): top에 있는 원소를 반환하고 삭제
- stack[-1]: top에 있는 원소
- len(stack): 스택 사이즈를 반환
- len(stack) == 0: 스택이 비어있는지 확인



서비스 접속 대기 중입니다.

예상대기시간 : 1초

앞에 1 명, 뒤에 1 명의 대기자가 있습니다.  
현재 접속 사용자가 많아 대기 중이며, 잠시만 기다리시면  
서비스로 자동 접속 됩니다.

※ 재 접속하시면 대기시간이 더 길어집니다. [중지]

<50 ~ 54세 연령층(1967.1.1. ~ 1971.12.31. 출생)>

○ (예약기간)

53~54세(67~68년생) : 7.19.(월) 20시 ~ 7.20.(화) 18시

50~52세(69~71년생) : 7.20.(화) 20시 ~ 7.21.(수) 18시

50~54세(67~71년생) : 7.21.(수) 20시 ~ 7.24.(토) 18시

\* 예방접종센터 예약은 7.21.(수) 20시 ~ 7.24.(토) 18시

○ (접종기간) 8.16.(월) ~ 8.28.(토)

<60 ~ 74세 고령층 중 사전예약 후 미접종자(1947.1.1. ~ 1961.12.31. 출생)>

○ (예약기간) 7.14.(수) 20시 ~ 7.24.(토) 18시

○ (접종기간) 7.26.(월) ~

서비스 접속 대기 중입니다.

예상대기시간 : 3시간 41분 19초

고객님 앞에 172442 명, 뒤에 7713 명의 대기자가 있습니다.  
현재 접속 사용자가 많아 대기 중이며, 잠시만 기다리시면  
서비스로 자동 접속 됩니다.

※ 재 접속하시면 대기시간이 더 길어집니다. [중지]

코로나19 예방접종 사전예약 시스템 | 개인정보처리방침 | 질병관리청 바로가기

(28159) 충북 청주시 흥덕구 오송읍 오송생명2로 187 오송보건의료행정타운내 질병관리청

COPYRIGHT © 2021 질병관리청 ALL RIGHTS RESERVED.

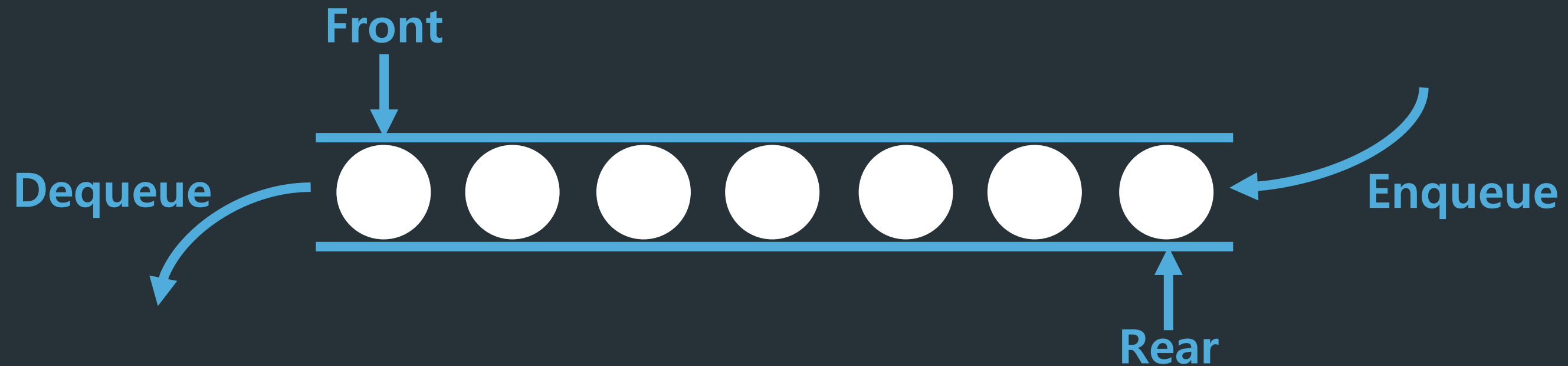


와일드 정글

기존의 시시함은 가라! 이것이 리얼 어드벤처!

110cm 미만

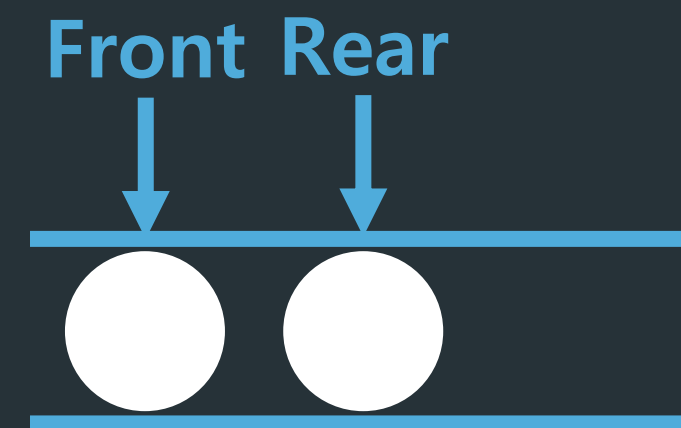
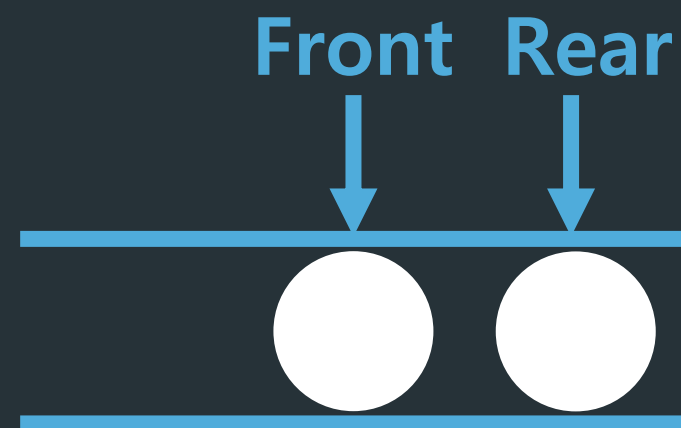
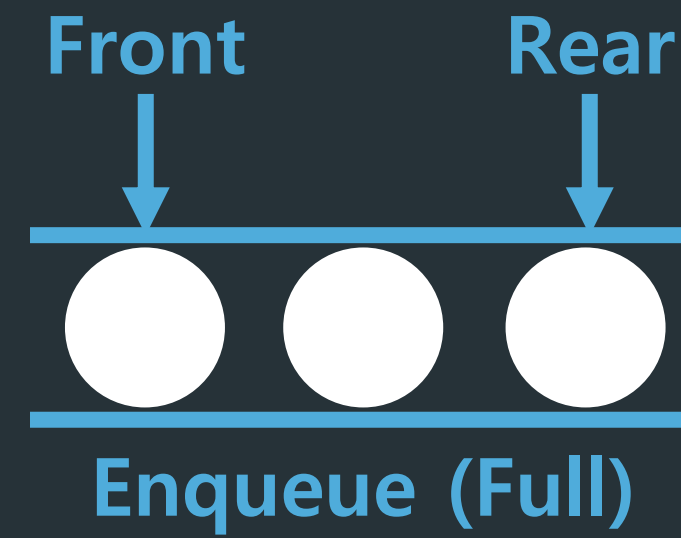
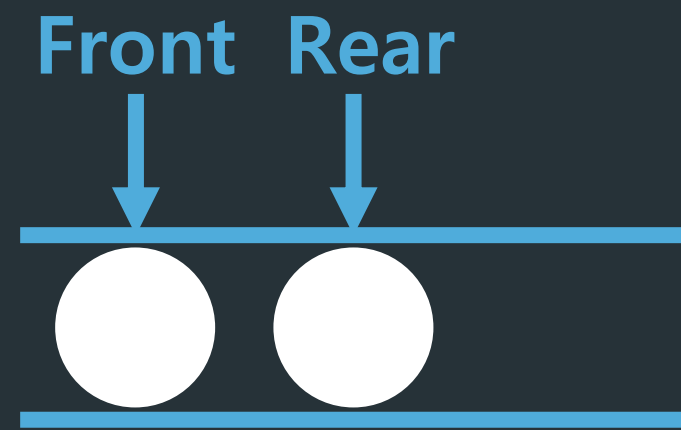
현장대기  
45분



## Queue

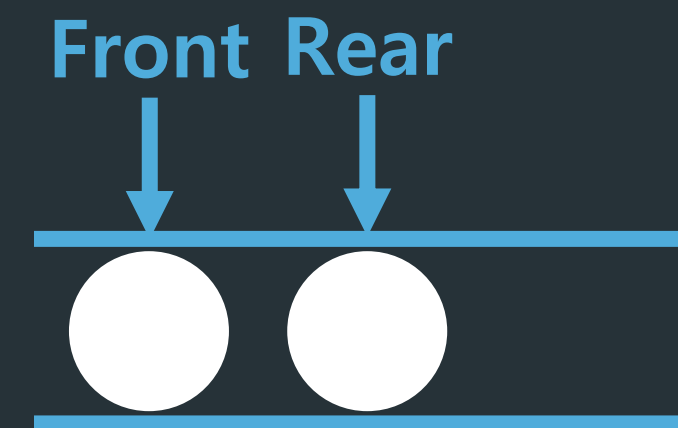
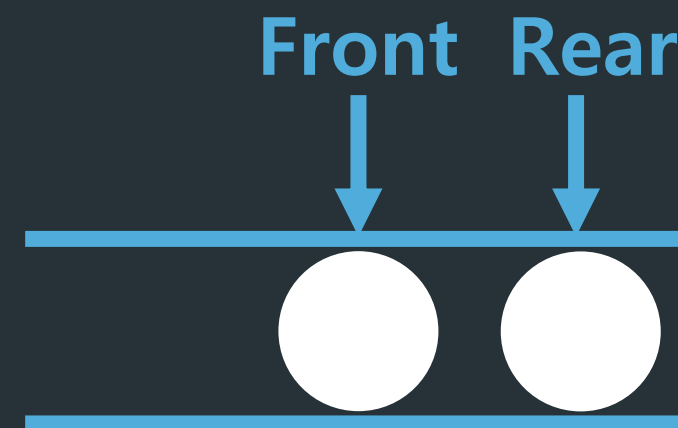
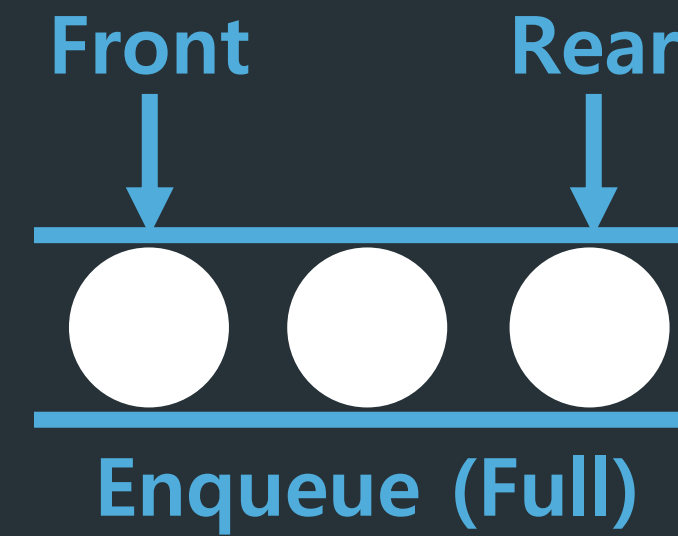
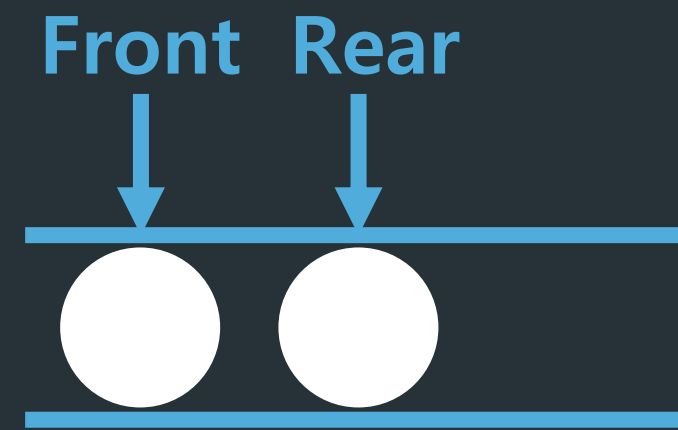
- FIFO (First In First Out)
- 자료의 왼쪽 끝 위치에서 삭제, 오른쪽 끝 위치에서 삽입 연산이 이루어짐
- 모든 연산에 대한 시간 복잡도는  $O(1)$
- 삭제가 이루어지는 위치를 *front*, 삽입이 이루어지는 위치를 *rear*라고 부름
- 삭제는 *dequeue*, 삽입은 *enqueue*

# 배열로 크기 3인 큐 구현하기?



Dequeue?

# 배열로 크기 3인 큐 구현하기?



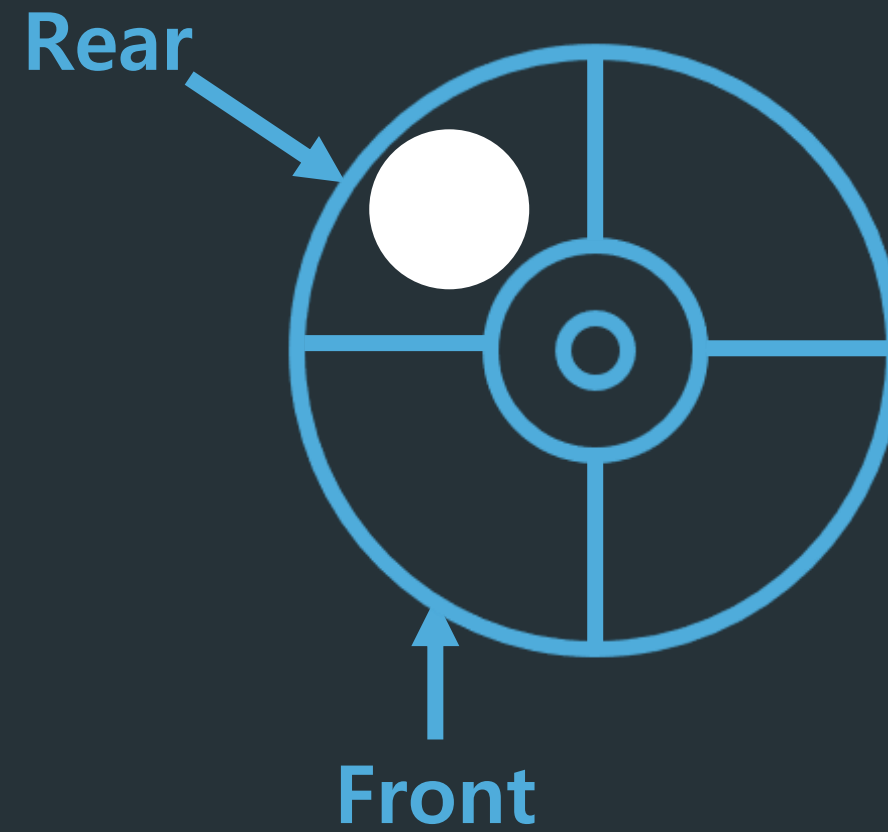
Dequeue?

Dequeue 연산마다 배열의 모든 원소를 한 칸씩 옮기는 건 비효율적이다!

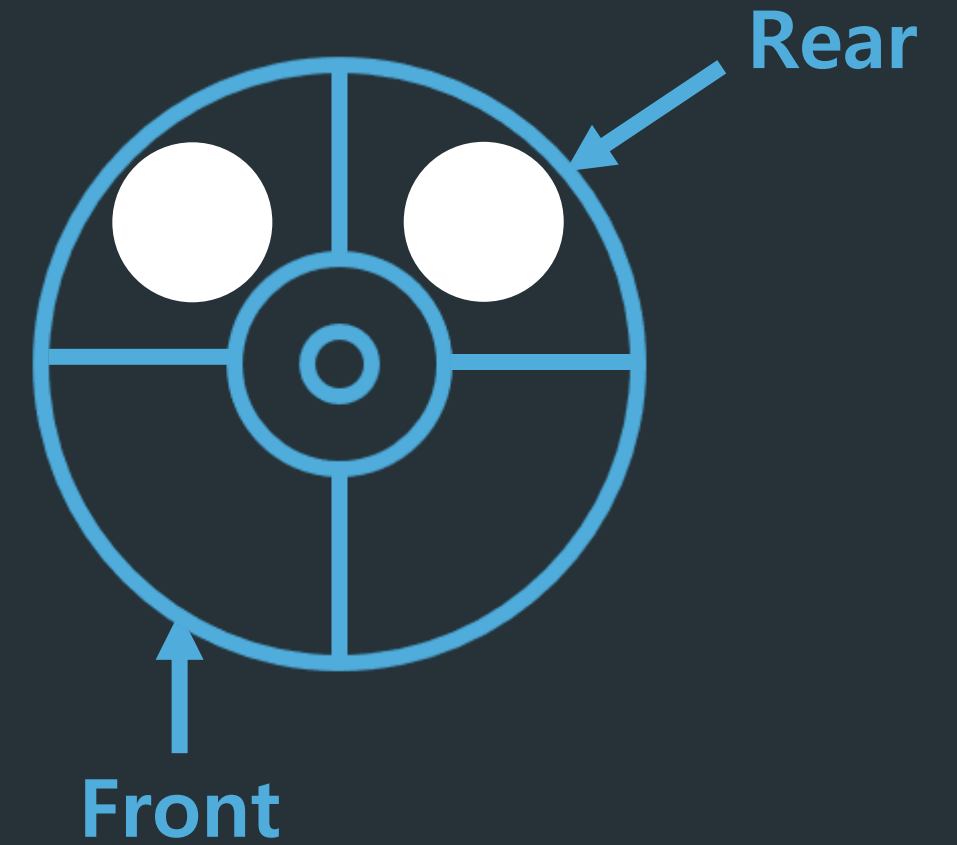
# 배열로 크기 3인 큐 구현하기



Empty



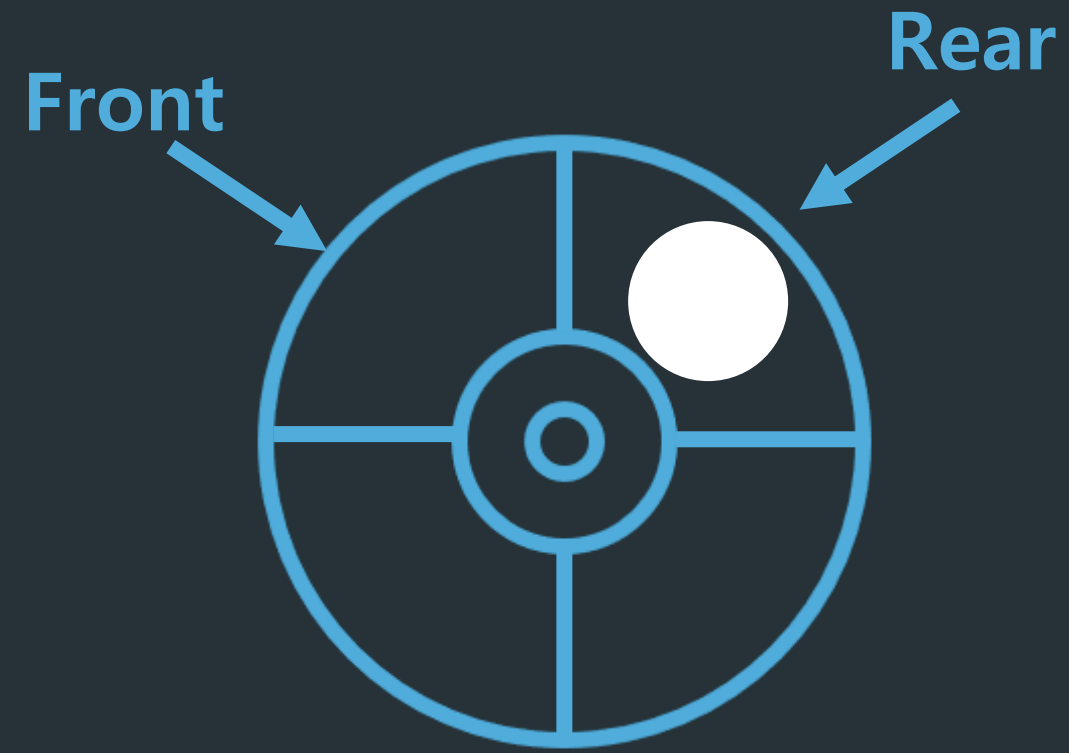
Enqueue



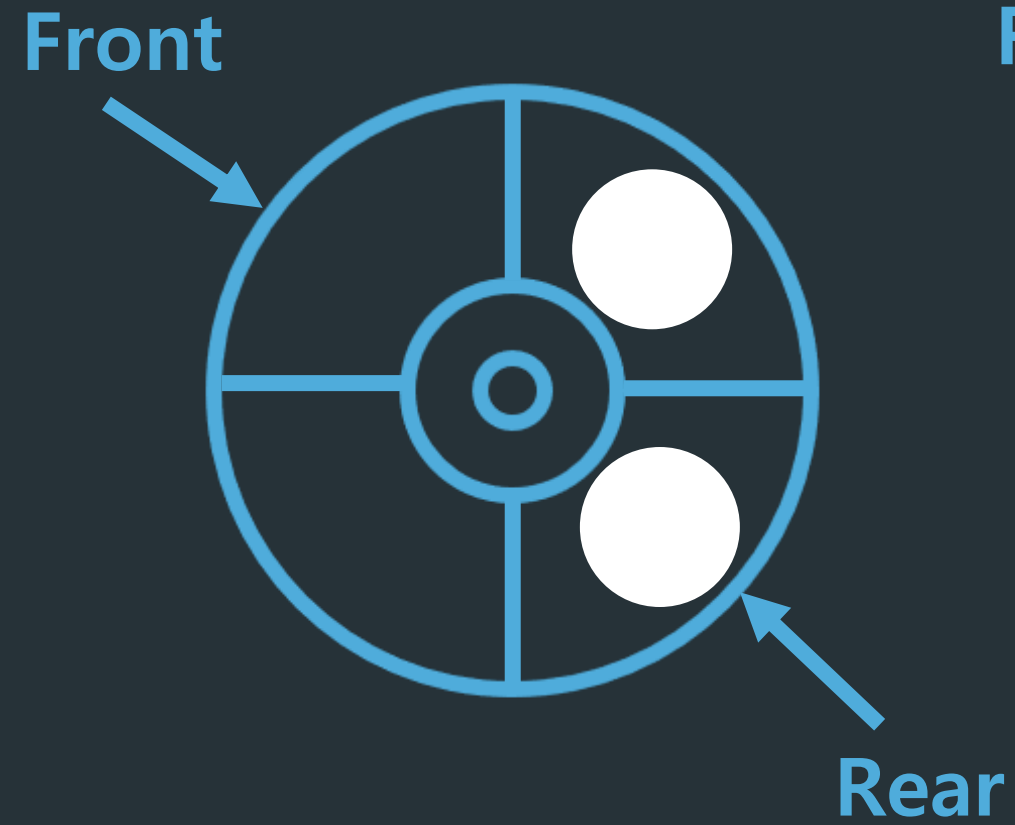
Enqueue

Rear == Front : Empty

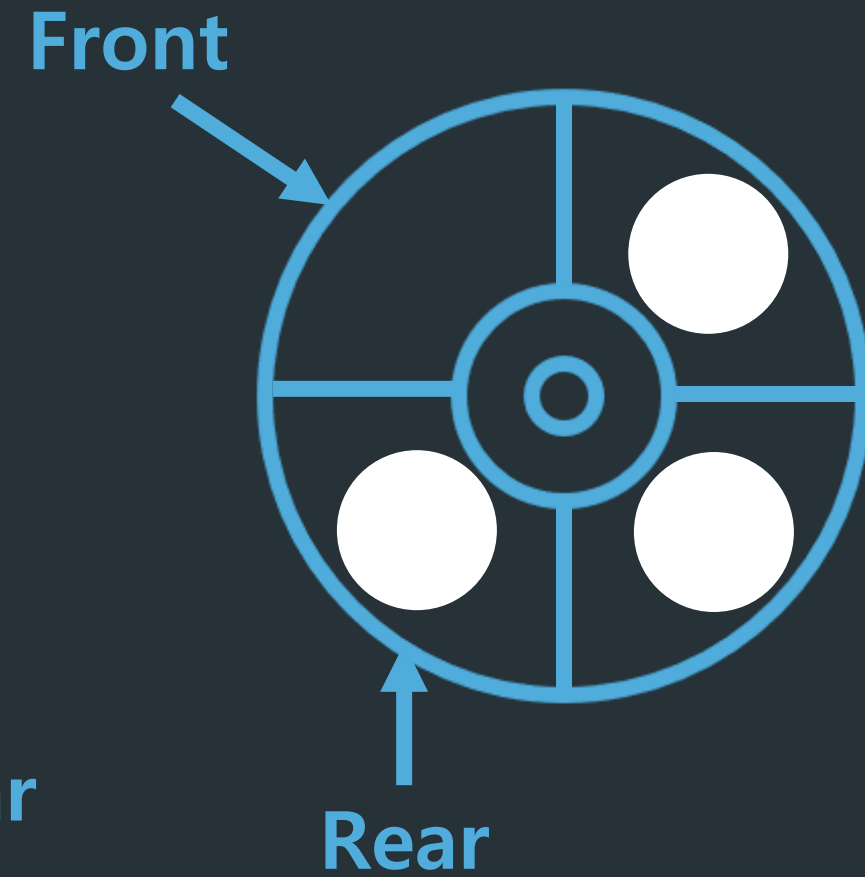
# 배열로 크기 3인 큐 구현하기



Dequeue



Enqueue



Enqueue (Full)

$(Rear + 1) \% SIZE == Front$  : Full



## /<> 10845번 : 큐 – Silver 4

### 문제

- 다음의 명령을 처리하는 큐 프로그램 만들기
  1. push X : 정수 X를 큐에 **삽입**
  2. pop : 큐에서 가장 앞에 있는 정수를 **빼고**, 출력. 큐가 비었다면 -1 출력
  3. size : 큐에 들어있는 정수의 **개수** 출력
  4. empty : 큐가 **비었으면 1, 아니라면 0**을 출력
  5. front : 큐의 가장 앞에 있는 정수를 **출력**. 큐가 비었다면 -1 출력
  6. back : 큐의 가장 뒤에 있는 정수를 **출력**. 큐가 비었다면 -1 출력

### 제한 사항

- 명령의 수 N의 범위는  $1 \leq N \leq 10,000$
- 명령과 함께 주어지는 정수 k의 범위는  $1 \leq k \leq 100,000$


## 예제 입력1

```
15
push 1
push 2
front
back
size
empty
pop
pop
pop
size
empty
pop
push 3
empty
front
```

## 예제 출력1

```
1
2
2
0
1
2
-1
0
1
-1
0
3
```

# 큐를 쓸 일도 정말 많죠...!



Search:  Go

Not logged in

register log in

Reference <queue> queue

You were redirected to cplusplus.com/queue || See search results for: "queue"

C++

Information

Tutorials

Reference

Articles

Forum

Reference

C library:

Containers:

<array>

<deque>

<forward\_list>

<list>

<map>

<queue>

<set>

<stack>

<unordered\_map>

<unordered\_set>

<vector>

Input/Output:

Multi-threading:

Other:

<queue>

priority\_queue

queue

queue

queue::queue

member functions:

queue::back

queue::emplace

queue::empty

queue::front

queue::pop

queue::push

queue::size

class template

std::queue

template &lt;class T, class Container = deque&lt;T&gt; &gt; class queue;

FIFO queue

queues are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

queues are implemented as *containers adaptors*, which are classes that use an encapsulated object of a specific container class as its *underlying container*, providing a specific set of member functions to access its elements. Elements are *pushed* into the "back" of the specific container and *popped* from its "front".

The underlying container may be one of the standard container class template or some other specifically designed container class. This underlying container shall support at least the following operations:

- empty
- size
- front
- back
- push\_back
- pop\_front

The standard container classes `deque` and `list` fulfill these requirements. By default, if no container class is specified for a particular queue class instantiation, the standard container `deque` is used.

Template parameters

T

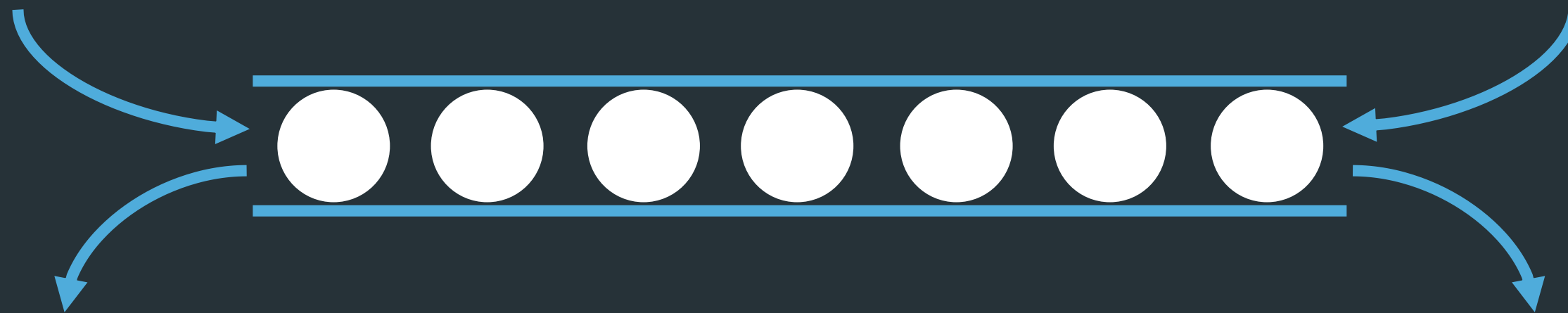
Type of the elements.  
Aliased as member type `queue::value_type`.

Container

Type of the internal *underlying container* object where the elements are stored.  
Its `value_type` shall be T.  
Aliased as member type `queue::container_type`.

## std::queue

- push(element): 큐의 뒤에 원소를 추가
- pop(): 큐의 앞에 있는 원소를 삭제
- front(): 큐의 제일 앞에 있는 원소를 반환
- back(): 큐의 제일 뒤에 있는 원소를 반환
- empty(): 큐가 비어있는지 확인 (비어있으면 true)
- size(): 큐의 사이즈를 반환



## Deque

- Double-Ended Queue
- Stack + Queue
- 자료의 양 끝에서 연산이 이루어짐
- 모든 연산에 대한 시간 복잡도는  $O(1)$

## std::deque

- push\_front(element): 덱의 앞에 원소를 추가
- push\_back(element): 덱의 뒤에 원소를 추가
- pop\_front(): 덱의 앞에 있는 원소를 삭제
- pop\_back(): 덱의 뒤에 있는 원소를 삭제
- front(): 덱의 제일 앞에 있는 원소를 반환
- back(): 덱의 제일 뒤에 있는 원소를 반환
- empty(): 덱이 비어있는지 확인 (비어있으면 true)
- size(): 덱의 사이즈를 반환

## deque objects

```
class collections.deque([iterable[, maxlen]])
```

Returns a new deque object initialized left-to-right (using `append()`) with data from *iterable*. If *iterable* is not specified, the new deque is empty.

Deques are a generalization of stacks and queues (the name is pronounced “deck” and is short for “double-ended queue”). Deques support thread-safe, memory efficient appends and pops from either side of the deque with approximately the same  $O(1)$  performance in either direction.

Though `list` objects support similar operations, they are optimized for fast fixed-length operations and incur  $O(n)$  memory movement costs for `pop(0)` and `insert(0, v)` operations which change both the size and position of the underlying data representation.

If *maxlen* is not specified or is `None`, deques may grow to an arbitrary length. Otherwise, the deque is bounded to the specified maximum length. Once a bounded length deque is full, when new items are added, a corresponding number of items are discarded from the opposite end. Bounded length deques provide functionality similar to the `tail` filter in Unix. They are also useful for tracking transactions and other pools of data where only the most recent activity is of interest.

```
import collections
deq = collections.deque()
```

or

```
from collections import deque
deq = deque()
```



## collections.deque

- `deq = collections.deque()` # 덱을 정의
- `.appendleft(element)`: 덱의 앞에 원소를 추가
- `.append(element)`: 덱의 뒤에 원소를 추가
- `.popleft()`: 덱의 앞에 있는 원소를 반환하고 삭제
- `.pop()`: 덱의 뒤에 있는 원소를 반환하고 삭제
- `deq[0]`: 덱의 제일 앞에 있는 원소
- `deq[-1]`: 덱의 제일 뒤에 있는 원소
- `len(deq)`: 덱의 사이즈를 반환
- `len(deq)==0`: 덱이 비어있는지 확인

`queue.Queue`가 따로 있지만, 이는 스레드 프로그래밍을 위한 것으로 일반적으로 `collections.deque`보다 느리다!

따라서, `collections.deque`의 `.append()`와 `.popleft()`로 큐를 사용하자.

## /<> 10866번 : 덱 – Silver 4

### 문제

- 다음의 명령을 처리하는 덱 프로그램 만들기
  1. push\_front X : 정수 X를 덱의 앞에 삽입
  2. push\_back X : 정수 X를 덱의 뒤에 삽입
  3. pop\_front : 덱에서 가장 앞에 있는 정수를 빼고, 출력. 덱이 비었다면 -1 출력
  4. pop\_back : 덱에서 가장 뒤에 있는 정수를 빼고, 출력. 덱이 비었다면 -1 출력
  5. size : 덱에 들어있는 정수의 개수 출력
  6. empty : 덱이 비었으면 1, 아니라면 0을 출력
  7. front : 덱의 가장 앞에 있는 정수를 출력. 덱이 비었다면 -1 출력
  8. back : 덱의 가장 뒤에 있는 정수를 출력. 덱이 비었다면 -1 출력

### 제한 사항

- 명령의 수 N의 범위는  $1 \leq N \leq 10,000$
- 명령과 함께 주어지는 정수 k의 범위는  $1 \leq k \leq 100,000$

## 예제 입력1

```
15
push_back 1
push_front 2
front
back
size
empty
pop_front
pop_back
pop_front
size
empty
pop_back
push_front 3
empty
front
```

## 예제 출력1

```
2
1
2
0
2
1
-1
0
1
-1
0
3
```

## /<> 4949번 : 균형잡힌 세상 – Silver 4

### 문제

- 문자열이 주어졌을 때, 괄호의 짝이 잘 맞는지 판단

### 제한 사항

- 문자열의 길이는 100글자보다 작거나 같다

## 예제 입력1

```
So when I die (the [first] I will see in (heaven) is a score list).  
[ first in ] ( first out ).  
Half Moon tonight (At least it is better than no Moon at all].  
A rope may form )( a trail in a maze.  
Help( I[m being held prisoner in a fortune cookie factory)].  
([ (([ ( [ ] ) ( ) (( )) ] )) ).  
.  
.
```

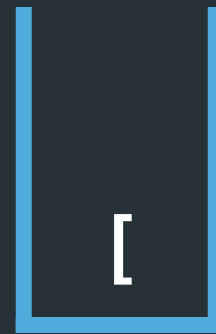
## 예제 출력1

```
yes  
yes  
no  
no  
no  
yes  
yes
```

## Hint

1. 띄어쓰기까지 포함해서 입력을 받으려면 어떻게 해야 할까요?
2. ((([ ])))

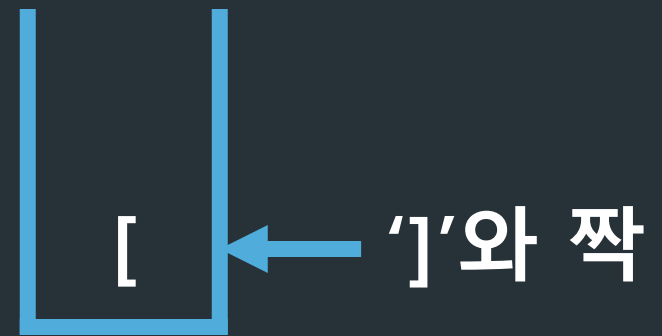
# 괄호의 짝이 맞는 경우



[ first in ] ( first out ).

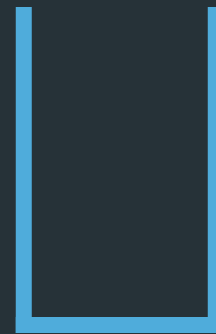


# 괄호의 짝이 맞는 경우



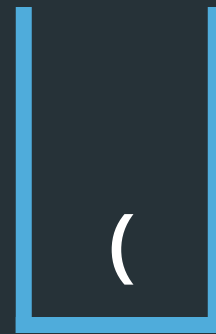
[ first in ] ( first out ).

# | 괄호의 짝이 맞는 경우



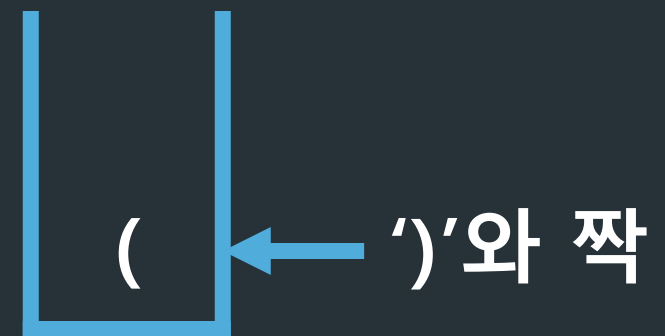
[ first in ] ( first out ).

# | 괄호의 짝이 맞는 경우



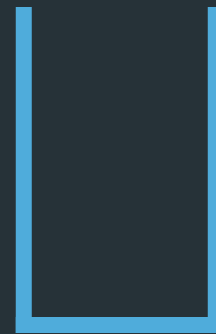
[ first in ] ( first out ).

# 괄호의 짝이 맞는 경우



[ first in ] ( first out ).

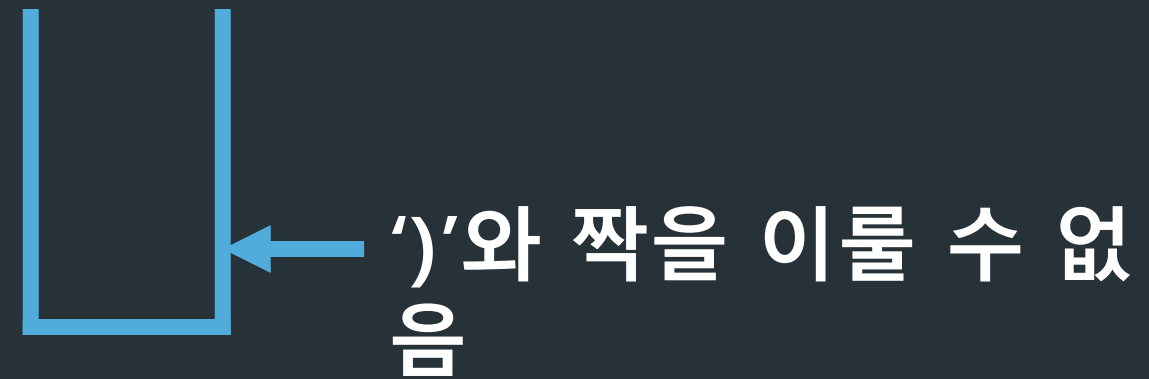
# 괄호의 짝이 맞는 경우



[ first in ] ( first out ).

Empty Stack

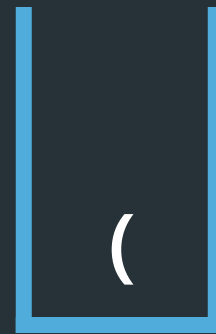
# 괄호의 짝이 맞지 않는 경우 1



A rope may form )( a trail in a maze

Empty Stack

## 괄호의 짝이 맞지 않는 경우 2



Help( I[m being held prisoner in a fortune cookie factory)].



## 괄호의 짝이 맞지 않는 경우 2

[  
(

Help( I[m being held prisoner in a fortune cookie factory)].

## 괄호의 짝이 맞지 않는 경우 2

[  
( ← ')'와 짝을 이룰 수 없음

Help( I[m being held prisoner in a fortune cookie factory)].

# | 괄호의 짝이 맞지 않는 경우 3



(

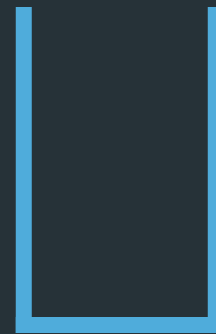
( ) (

# 괄호의 짝이 맞지 않는 경우 3

( ← ')'와 짝

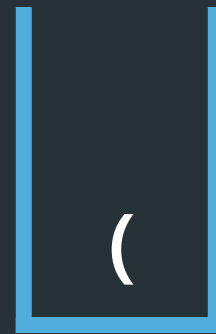
( ) (

# | 괄호의 짝이 맞지 않는 경우 3



( ) (

# | 괄호의 짝이 맞지 않는 경우 3



() (

Not Empty

## 정리

- 스택, 큐, 덱 모두 연산에서의 시간 복잡도가  $O(1)$ 인 자료구조
- 효율성을 보는 문제에 사용되는 경우가 많음
- 순회는 벡터보다 불편함
- 무한 루프 (pop을 하지 않음), 런타임 에러 (empty 체크 안하고 조회 or 삭제 시도) 조심!!

## 이것도 알아보세요!

- 재귀 함수로 짠 알고리즘은 스택으로 구현할 수도 있는 경우가 많아요. 예습 할 겸 찾아보세요!

## 필수

- /<> 10757번 : 큰 수 A+B – Bronze 4
- /<> 2164번 : 카드2 – Silver 4
- /<> 4949번 : 균형잡힌 세상 – Silver 4

## 도전

- /<> 17298 : 오큰수 – Gold 4
- /<> 1918번 : 후위 표기식 – Gold 2