Problem 1845: Seat Reservation Manager

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Design a system that manages the reservation state of

n

seats that are numbered from

1

to

n

.

Implement the

SeatManager

class:

SeatManager(int n)

Initializes a

SeatManager

object that will manage

n

seats numbered from

1

to

n

. All seats are initially available.

int reserve()

Fetches the

smallest-numbered

unreserved seat, reserves it, and returns its number.

void unreserve(int seatNumber)

Unreserves the seat with the given

seatNumber

.

Example 1:

Input

["SeatManager", "reserve", "reserve", "unreserve", "reserve", "reserve", "reserve", "reserve", "unreserve"] [[5], [], [], [2], [], [], [], [], [5]]

Output

[null, 1, 2, null, 2, 3, 4, 5, null]

Explanation

SeatManager seatManager = new SeatManager(5); // Initializes a SeatManager with 5 seats. seatManager.reserve(); // All seats are available, so return the lowest numbered seat, which is 1. seatManager.reserve(); // The available seats are [2,3,4,5], so return the lowest of them, which is 2. seatManager.unreserve(2); // Unreserve seat 2, so now the available seats are [2,3,4,5]. seatManager.reserve(); // The available seats are [2,3,4,5], so return the lowest of them, which is 2. seatManager.reserve(); // The available seats are [3,4,5], so return the lowest of them, which is 3. seatManager.reserve(); // The available seats are [4,5], so return the lowest of them, which is 4. seatManager.reserve(); // The only available seat is seat 5, so return 5. seatManager.unreserve(5); // Unreserve seat 5, so now the available seats are [5].

Constraints:

1 <= n <= 10

5

1 <= seatNumber <= n

For each call to

reserve

, it is guaranteed that there will be at least one unreserved seat.

For each call to

unreserve

, it is guaranteed that

seatNumber

will be reserved.

At most

10

5

calls

in total

will be made to

reserve

and

unreserve

.

Code Snippets

C++:

```
class SeatManager {
public:
SeatManager(int n) {

}

int reserve() {

}

void unreserve(int seatNumber) {

}
};

/**
* Your SeatManager object will be instantiated and called as such:
```

```
* SeatManager* obj = new SeatManager(n);
* int param_1 = obj->reserve();
* obj->unreserve(seatNumber);
*/
```

Java:

```
class SeatManager {

public SeatManager(int n) {

}

public int reserve() {

}

public void unreserve(int seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = new SeatManager(n);
* int param_1 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Python3:

```
class SeatManager:

def __init__(self, n: int):


def reserve(self) -> int:


def unreserve(self, seatNumber: int) -> None:
```

```
# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager(n)
# param_1 = obj.reserve()
# obj.unreserve(seatNumber)
```

Python:

```
class SeatManager(object):

def __init__(self, n):
"""
:type n: int
"""



def reserve(self):
"""
:rtype: int
"""



def unreserve(self, seatNumber):
"""
:type seatNumber: int
:rtype: None
"""




# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager(n)
# param_1 = obj.reserve()
# obj.unreserve(seatNumber)
```

JavaScript:

```
/**
* @param {number} n
*/
var SeatManager = function(n) {
```

```
};

/**
* @return {number}
*/
SeatManager.prototype.reserve = function() {

};

/**
* @param {number} seatNumber
* @return {void}
*/
SeatManager.prototype.unreserve = function(seatNumber) {

};

/**
* Your SeatManager object will be instantiated and called as such:
* var obj = new SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

TypeScript:

```
class SeatManager {
constructor(n: number) {

}

reserve(): number {

}

unreserve(seatNumber: number): void {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
```

```
* var obj = new SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

C#:

```
public class SeatManager {

public SeatManager(int n) {

}

public int Reserve() {

}

public void Unreserve(int seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = new SeatManager(n);
* int param_1 = obj.Reserve();
* obj.Unreserve(seatNumber);
*/
```

C:

```
typedef struct {

} SeatManager;


SeatManager* seatManagerCreate(int n) {

}
```

```
int seatManagerReserve(SeatManager* obj) {

}

void seatManagerUnreserve(SeatManager* obj, int seatNumber) {

}

void seatManagerFree(SeatManager* obj) {

}

/**
* Your SeatManager struct will be instantiated and called as such:
* SeatManager* obj = seatManagerCreate(n);
* int param_1 = seatManagerReserve(obj);

* seatManagerUnreserve(obj, seatNumber);

* seatManagerFree(obj);
*/
```

Go:

```
type SeatManager struct {

}


func Constructor(n int) SeatManager {

}


func (this *SeatManager) Reserve() int {

}


func (this *SeatManager) Unreserve(seatNumber int) {
```

```
}


/**
* Your SeatManager object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Reserve();
* obj.Unreserve(seatNumber);
*/
```

Kotlin:

```
class SeatManager(n: Int) {

fun reserve(): Int {

}

fun unreserve(seatNumber: Int) {

}

}

/**
* Your SeatManager object will be instantiated and called as such:
* var obj = SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Swift:

```


class SeatManager {

init(_ n: Int) {

}

func reserve() -> Int {
```

```
}

func unreserve(_ seatNumber: Int) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* let obj = SeatManager(n)
* let ret_1: Int = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Rust:

```
struct SeatManager {

}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SeatManager {

fn new(n: i32) -> Self {

}

fn reserve(&self) -> i32 {

}

fn unreserve(&self, seat_number: i32) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
```

```
* let obj = SeatManager::new(n);
* let ret_1: i32 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Ruby:

```
class SeatManager

=begin
:type n: Integer
=end
def initialize(n)

end


=begin
:rtype: Integer
=end
def reserve()

end


=begin
:type seat_number: Integer
:rtype: Void
=end
def unreserve(seat_number)

end


end

# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager.new(n)
# param_1 = obj.reserve()
# obj.unreserve(seat_number)
```

PHP:

```
class SeatManager {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @return Integer
*/
function reserve() {

}

/**
* @param Integer $seatNumber
* @return NULL
*/
function unreserve($seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* $obj = SeatManager($n);
* $ret_1 = $obj->reserve();
* $obj->unreserve($seatNumber);
*/
```

Dart:

```
class SeatManager {

SeatManager(int n) {

}

int reserve() {
```

```
}

void unreserve(int seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = SeatManager(n);
* int param1 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Scala:

```
class SeatManager(_n: Int) {

def reserve(): Int = {

}

def unreserve(seatNumber: Int): Unit = {

}

}

/**
* Your SeatManager object will be instantiated and called as such:
* val obj = new SeatManager(n)
* val param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Elixir:

```
defmodule SeatManager do
@spec init_(n :: integer) :: any
def init_(n) do

end
```

```
@spec reserve() :: integer
def reserve() do

end

@spec unreserve(seat_number :: integer) :: any
def unreserve(seat_number) do

end
end

# Your functions will be called as such:
# SeatManager.init_(n)
# param_1 = SeatManager.reserve()
# SeatManager.unreserve(seat_number)

# SeatManager.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec seat_manager_init_(N :: integer()) -> any().
seat_manager_init_(N) ->
.

-spec seat_manager_reserve() -> integer().
seat_manager_reserve() ->
.

-spec seat_manager_unreserve(SeatNumber :: integer()) -> any().
seat_manager_unreserve(SeatNumber) ->
.


%% Your functions will be called as such:
%% seat_manager_init_(N),
%% Param_1 = seat_manager_reserve(),
%% seat_manager_unreserve(SeatNumber),

%% seat_manager_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket:

```
(define seat-manager%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; reserve : -> exact-integer?
(define/public (reserve)
)
; unreserve : exact-integer? -> void?
(define/public (unreserve seat-number)
)))

;; Your seat-manager% object will be instantiated and called as such:
;; (define obj (new seat-manager% [n n]))
;; (define param_1 (send obj reserve))
;; (send obj unreserve seat-number)
```

Solutions

C++ Solution:

```
/*
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class SeatManager {
public:
SeatManager(int n) {

}
```

```
int reserve() {

}

void unreserve(int seatNumber) {

}
};

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager* obj = new SeatManager(n);
* int param_1 = obj->reserve();
* obj->unreserve(seatNumber);
*/
```

Java Solution:

```
/**
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class SeatManager {

public SeatManager(int n) {

}

public int reserve() {

}

public void unreserve(int seatNumber) {
```

```
}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = new SeatManager(n);
* int param_1 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Python3 Solution:

```
"""
Problem: Seat Reservation Manager
Difficulty: Medium
Tags: queue, heap

Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class SeatManager:

def __init__(self, n: int):


def reserve(self) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class SeatManager(object):

def __init__(self, n):
"""
:type n: int
"""
```

```
def reserve(self):
"""
:rtype: int
"""


def unreserve(self, seatNumber):
"""
:type seatNumber: int
:rtype: None
"""



# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager(n)
# param_1 = obj.reserve()
# obj.unreserve(seatNumber)
```

JavaScript Solution:

```
/**
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
*/
var SeatManager = function(n) {

};

/**
* @return {number}
*/
```

```
SeatManager.prototype.reserve = function() {

};

/**
* @param {number} seatNumber
* @return {void}
*/
SeatManager.prototype.unreserve = function(seatNumber) {

};

/**
* Your SeatManager object will be instantiated and called as such:
* var obj = new SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

TypeScript Solution:

```
/**
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class SeatManager {
constructor(n: number) {

}

reserve(): number {

}

unreserve(seatNumber: number): void {
```

```
}
}

/**
* Your SeatManager object will be instantiated and called as such:
* var obj = new SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

C# Solution:

```
/*
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class SeatManager {

public SeatManager(int n) {

}

public int Reserve() {

}

public void Unreserve(int seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = new SeatManager(n);
```

```
* int param_1 = obj.Reserve();
* obj.Unreserve(seatNumber);
*/
```

C Solution:

```
/*
* Problem: Seat Reservation Manager
* Difficulty: Medium
* Tags: queue, heap
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} SeatManager;


SeatManager* seatManagerCreate(int n) {

}

int seatManagerReserve(SeatManager* obj) {

}

void seatManagerUnreserve(SeatManager* obj, int seatNumber) {

}

void seatManagerFree(SeatManager* obj) {

}

/**
```

```
* Your SeatManager struct will be instantiated and called as such:
* SeatManager* obj = seatManagerCreate(n);
* int param_1 = seatManagerReserve(obj);

* seatManagerUnreserve(obj, seatNumber);

* seatManagerFree(obj);
*/
```

Go Solution:

```
// Problem: Seat Reservation Manager
// Difficulty: Medium
// Tags: queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

type SeatManager struct {

}


func Constructor(n int) SeatManager {

}


func (this *SeatManager) Reserve() int {

}


func (this *SeatManager) Unreserve(seatNumber int) {

}


/**
* Your SeatManager object will be instantiated and called as such:
```

```
* obj := Constructor(n);
* param_1 := obj.Reserve();
* obj.Unreserve(seatNumber);
*/
```

Kotlin Solution:

```
class SeatManager(n: Int) {

fun reserve(): Int {

}

fun unreserve(seatNumber: Int) {

}

}

/**
* Your SeatManager object will be instantiated and called as such:
* var obj = SeatManager(n)
* var param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Swift Solution:

```
class SeatManager {

init(_ n: Int) {

}

func reserve() -> Int {

}

func unreserve(_ seatNumber: Int) {
```

```
}
}

/**
* Your SeatManager object will be instantiated and called as such:
* let obj = SeatManager(n)
* let ret_1: Int = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Rust Solution:

```
// Problem: Seat Reservation Manager
// Difficulty: Medium
// Tags: queue, heap
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

struct SeatManager {

}



/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl SeatManager {

fn new(n: i32) -> Self {

}

fn reserve(&self) -> i32 {

}

fn unreserve(&self, seat_number: i32) {
```

```
}
}

/**
* Your SeatManager object will be instantiated and called as such:
* let obj = SeatManager::new(n);
* let ret_1: i32 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Ruby Solution:

```
class SeatManager

=begin
:type n: Integer
=end
def initialize(n)

end


=begin
:rtype: Integer
=end
def reserve()

end


=begin
:type seat_number: Integer
:rtype: Void
=end
def unreserve(seat_number)

end


end
```

```
# Your SeatManager object will be instantiated and called as such:
# obj = SeatManager.new(n)
# param_1 = obj.reserve()
# obj.unreserve(seat_number)
```

PHP Solution:

```
class SeatManager {
/**
* @param Integer $n
*/
function __construct($n) {

}

/**
* @return Integer
*/
function reserve() {

}

/**
* @param Integer $seatNumber
* @return NULL
*/
function unreserve($seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* $obj = SeatManager($n);
* $ret_1 = $obj->reserve();
* $obj->unreserve($seatNumber);
*/
```

Dart Solution:

```
class SeatManager {

SeatManager(int n) {

}

int reserve() {

}

void unreserve(int seatNumber) {

}
}

/**
* Your SeatManager object will be instantiated and called as such:
* SeatManager obj = SeatManager(n);
* int param1 = obj.reserve();
* obj.unreserve(seatNumber);
*/
```

Scala Solution:

```
class SeatManager(_n: Int) {

def reserve(): Int = {

}

def unreserve(seatNumber: Int): Unit = {

}

}

/**
* Your SeatManager object will be instantiated and called as such:
* val obj = new SeatManager(n)
* val param_1 = obj.reserve()
* obj.unreserve(seatNumber)
*/
```

Elixir Solution:

```
defmodule SeatManager do
@spec init_(n :: integer) :: any
def init_(n) do

end

@spec reserve() :: integer
def reserve() do

end

@spec unreserve(seat_number :: integer) :: any
def unreserve(seat_number) do

end
end

# Your functions will be called as such:
# SeatManager.init_(n)
# param_1 = SeatManager.reserve()
# SeatManager.unreserve(seat_number)

# SeatManager.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec seat_manager_init_(N :: integer()) -> any().
seat_manager_init_(N) ->
.

-spec seat_manager_reserve() -> integer().
seat_manager_reserve() ->
.

-spec seat_manager_unreserve(SeatNumber :: integer()) -> any().
seat_manager_unreserve(SeatNumber) ->
.
```

```
%% Your functions will be called as such:
%% seat_manager_init_(N),
%% Param_1 = seat_manager_reserve(),
%% seat_manager_unreserve(SeatNumber),

%% seat_manager_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
(define seat-manager%
(class object%
(super-new)

; n : exact-integer?
(init-field
n)

; reserve : -> exact-integer?
(define/public (reserve)
)
; unreserve : exact-integer? -> void?
(define/public (unreserve seat-number)
)))

;; Your seat-manager% object will be instantiated and called as such:
;; (define obj (new seat-manager% [n n]))
;; (define param_1 (send obj reserve))
;; (send obj unreserve seat-number)
```