Problem 855: Exam Room

Problem Information

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

Problem Description

There is an exam room with

n

seats in a single row labeled from

0

to

n - 1

.

When a student enters the room, they must sit in the seat that maximizes the distance to the closest person. If there are multiple such seats, they sit in the seat with the lowest number. If no one is in the room, then the student sits at seat number

0

.

Design a class that simulates the mentioned exam room.

Implement the

ExamRoom

class:

ExamRoom(int n)

Initializes the object of the exam room with the number of the seats

n

.

int seat()

Returns the label of the seat at which the next student will set.

void leave(int p)

Indicates that the student sitting at seat

p

will leave the room. It is guaranteed that there will be a student sitting at seat

p

.

Example 1:

Input

["ExamRoom", "seat", "seat", "seat", "seat", "leave", "seat"] [[10], [], [], [], [], [4], []]

Output

[null, 0, 9, 4, 2, null, 5]

Explanation

ExamRoom examRoom = new ExamRoom(10); examRoom.seat(); // return 0, no one is in the room, then the student sits at seat number 0. examRoom.seat(); // return 9, the student sits at the last seat number 9. examRoom.seat(); // return 4, the student sits at the last seat number 4. examRoom.seat(); // return 2, the student sits at the last seat number 2. examRoom.leave(4); examRoom.seat(); // return 5, the student sits at the last seat number 5.

Constraints:

1 <= n <= 10

9

It is guaranteed that there is a student sitting at seat

p

.

At most

10

4

calls will be made to

seat

and

leave

.

Code Snippets

C++:

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

}

int seat() {

}

void leave(int p) {

}
};

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom* obj = new ExamRoom(n);
* int param_1 = obj->seat();
* obj->leave(p);
*/
```

Java:

```
class ExamRoom {

public ExamRoom(int n) {

}

public int seat() {

}

public void leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = new ExamRoom(n);
* int param_1 = obj.seat();
* obj.leave(p);
*/
```

Python3:

```
class ExamRoom:

def __init__(self, n: int):



def seat(self) -> int:



def leave(self, p: int) -> None:




# Your ExamRoom object will be instantiated and called as such:
# obj = ExamRoom(n)
# param_1 = obj.seat()
# obj.leave(p)
```

Python:

```
class ExamRoom(object):

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



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



def leave(self, p):
"""
:type p: int
:rtype: None
```

```
"""




# Your ExamRoom object will be instantiated and called as such:
# obj = ExamRoom(n)
# param_1 = obj.seat()
# obj.leave(p)
```

JavaScript:

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

};

/**
* @return {number}
*/
ExamRoom.prototype.seat = function() {

};

/**
* @param {number} p
* @return {void}
*/
ExamRoom.prototype.leave = function(p) {

};

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = new ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

TypeScript:

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

}

seat(): number {

}

leave(p: number): void {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = new ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

C#:

```
public class ExamRoom {

public ExamRoom(int n) {

}

public int Seat() {

}

public void Leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = new ExamRoom(n);
* int param_1 = obj.Seat();
* obj.Leave(p);
```

```
*/
```

C:

```



typedef struct {

} ExamRoom;


ExamRoom* examRoomCreate(int n) {

}

int examRoomSeat(ExamRoom* obj) {

}

void examRoomLeave(ExamRoom* obj, int p) {

}

void examRoomFree(ExamRoom* obj) {

}

/**
* Your ExamRoom struct will be instantiated and called as such:
* ExamRoom* obj = examRoomCreate(n);
* int param_1 = examRoomSeat(obj);

* examRoomLeave(obj, p);

* examRoomFree(obj);
*/
```

Go:

```
type ExamRoom struct {
}


func Constructor(n int) ExamRoom {

}


func (this *ExamRoom) Seat() int {

}


func (this *ExamRoom) Leave(p int) {

}


/**
* Your ExamRoom object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Seat();
* obj.Leave(p);
*/
```

Kotlin:

```
class ExamRoom(n: Int) {

fun seat(): Int {

}

fun leave(p: Int) {

}

}

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

Swift:

```
class ExamRoom {

init(_ n: Int) {

}

func seat() -> Int {

}

func leave(_ p: Int) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* let obj = ExamRoom(n)
* let ret_1: Int = obj.seat()
* obj.leave(p)
*/
```

Rust:

```
struct ExamRoom {

}



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

```
fn new(n: i32) -> Self {

}

fn seat(&self) -> i32 {

}

fn leave(&self, p: i32) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* let obj = ExamRoom::new(n);
* let ret_1: i32 = obj.seat();
* obj.leave(p);
*/
```

Ruby:

```
class ExamRoom

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

end



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

end



=begin
:type p: Integer
```

```
:rtype: Void
=end
def leave(p)

end


end

# Your ExamRoom object will be instantiated and called as such:
# obj = ExamRoom.new(n)
# param_1 = obj.seat()
# obj.leave(p)
```

PHP:

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

}

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

}

/**
* @param Integer $p
* @return NULL
*/
function leave($p) {

}
}

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

```
* $obj = ExamRoom($n);
* $ret_1 = $obj->seat();
* $obj->leave($p);
*/
```

Dart:

```
class ExamRoom {

ExamRoom(int n) {

}

int seat() {

}

void leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = ExamRoom(n);
* int param1 = obj.seat();
* obj.leave(p);
*/
```

Scala:

```
class ExamRoom(_n: Int) {

def seat(): Int = {

}

def leave(p: Int): Unit = {

}

}
```

```
/**
* Your ExamRoom object will be instantiated and called as such:
* val obj = new ExamRoom(n)
* val param_1 = obj.seat()
* obj.leave(p)
*/
```

Elixir:

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

end

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

end

@spec leave(p :: integer) :: any
def leave(p) do

end
end

# Your functions will be called as such:
# ExamRoom.init_(n)
# param_1 = ExamRoom.seat()
# ExamRoom.leave(p)

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

Erlang:

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

-spec exam_room_seat() -> integer().
```

```
exam_room_seat() ->
.

-spec exam_room_leave(P :: integer()) -> any().
exam_room_leave(P) ->
.


%% Your functions will be called as such:
%% exam_room_init_(N),
%% Param_1 = exam_room_seat(),
%% exam_room_leave(P),

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

Racket:

```
(define exam-room%
(class object%
(super-new)

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

; seat : -> exact-integer?
(define/public (seat)
)
; leave : exact-integer? -> void?
(define/public (leave p)
)))

;; Your exam-room% object will be instantiated and called as such:
;; (define obj (new exam-room% [n n]))
;; (define param_1 (send obj seat))
;; (send obj leave p)
```

Solutions

C++ Solution:

```
/*
* Problem: Exam Room
* 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 ExamRoom {
public:
ExamRoom(int n) {

}

int seat() {

}

void leave(int p) {

}
};

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom* obj = new ExamRoom(n);
* int param_1 = obj->seat();
* obj->leave(p);
*/
```

Java Solution:

```
/**
* Problem: Exam Room
* 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 ExamRoom {

public ExamRoom(int n) {

}

public int seat() {

}

public void leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = new ExamRoom(n);
* int param_1 = obj.seat();
* obj.leave(p);
*/
```

Python3 Solution:

```
"""
Problem: Exam Room
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 ExamRoom:

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

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

Python Solution:

```
class ExamRoom(object):

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



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



def leave(self, p):
"""
:type p: int
:rtype: None
"""




# Your ExamRoom object will be instantiated and called as such:
# obj = ExamRoom(n)
# param_1 = obj.seat()
# obj.leave(p)
```

JavaScript Solution:

```
/**
* Problem: Exam Room
* 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 ExamRoom = function(n) {

};

/**
* @return {number}
*/
ExamRoom.prototype.seat = function() {

};

/**
* @param {number} p
* @return {void}
*/
ExamRoom.prototype.leave = function(p) {

};

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = new ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

TypeScript Solution:

```
/**
* Problem: Exam Room
* 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 ExamRoom {
constructor(n: number) {

}

seat(): number {

}

leave(p: number): void {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = new ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

C# Solution:

```
/*
* Problem: Exam Room
* 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 ExamRoom {

public ExamRoom(int n) {
}

public int Seat() {

}

public void Leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = new ExamRoom(n);
* int param_1 = obj.Seat();
* obj.Leave(p);
*/
```

C Solution:

```
/*
* Problem: Exam Room
* 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 {

} ExamRoom;


ExamRoom* examRoomCreate(int n) {
}

int examRoomSeat(ExamRoom* obj) {

}

void examRoomLeave(ExamRoom* obj, int p) {

}

void examRoomFree(ExamRoom* obj) {

}

/**
* Your ExamRoom struct will be instantiated and called as such:
* ExamRoom* obj = examRoomCreate(n);
* int param_1 = examRoomSeat(obj);

* examRoomLeave(obj, p);

* examRoomFree(obj);
*/
```

Go Solution:

```
// Problem: Exam Room
// 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 ExamRoom struct {

}



func Constructor(n int) ExamRoom {
}


func (this *ExamRoom) Seat() int {

}


func (this *ExamRoom) Leave(p int) {

}


/**
* Your ExamRoom object will be instantiated and called as such:
* obj := Constructor(n);
* param_1 := obj.Seat();
* obj.Leave(p);
*/
```

Kotlin Solution:

```
class ExamRoom(n: Int) {

fun seat(): Int {

}

fun leave(p: Int) {

}

}

/**
* Your ExamRoom object will be instantiated and called as such:
* var obj = ExamRoom(n)
* var param_1 = obj.seat()
* obj.leave(p)
*/
```

Swift Solution:

```swift
class ExamRoom {

init(_ n: Int) {

}

func seat() -> Int {

}

func leave(_ p: Int) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* let obj = ExamRoom(n)
* let ret_1: Int = obj.seat()
* obj.leave(p)
*/
```

Rust Solution:

```rust
// Problem: Exam Room
// 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 ExamRoom {

}



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

```
*/
impl ExamRoom {

fn new(n: i32) -> Self {

}

fn seat(&self) -> i32 {

}

fn leave(&self, p: i32) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* let obj = ExamRoom::new(n);
* let ret_1: i32 = obj.seat();
* obj.leave(p);
*/
```

Ruby Solution:

```
class ExamRoom

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

end



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

end
```

```
=begin
:type p: Integer
:rtype: Void
=end
def leave(p)

end



end

# Your ExamRoom object will be instantiated and called as such:
# obj = ExamRoom.new(n)
# param_1 = obj.seat()
# obj.leave(p)
```

PHP Solution:

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

}

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

}

/**
* @param Integer $p
* @return NULL
*/
function leave($p) {
}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* $obj = ExamRoom($n);
* $ret_1 = $obj->seat();
* $obj->leave($p);
*/
```

Dart Solution:

```
class ExamRoom {

ExamRoom(int n) {

}

int seat() {

}

void leave(int p) {

}
}

/**
* Your ExamRoom object will be instantiated and called as such:
* ExamRoom obj = ExamRoom(n);
* int param1 = obj.seat();
* obj.leave(p);
*/
```

Scala Solution:

```
class ExamRoom(_n: Int) {

def seat(): Int = {

}
def leave(p: Int): Unit = {

}

}

/**
* Your ExamRoom object will be instantiated and called as such:
* val obj = new ExamRoom(n)
* val param_1 = obj.seat()
* obj.leave(p)
*/
```

Elixir Solution:

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

end

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

end

@spec leave(p :: integer) :: any
def leave(p) do

end
end

# Your functions will be called as such:
# ExamRoom.init_(n)
# param_1 = ExamRoom.seat()
# ExamRoom.leave(p)

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

Erlang Solution:

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

-spec exam_room_seat() -> integer().
exam_room_seat() ->
.

-spec exam_room_leave(P :: integer()) -> any().
exam_room_leave(P) ->
.



%% Your functions will be called as such:
%% exam_room_init_(N),
%% Param_1 = exam_room_seat(),
%% exam_room_leave(P),

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

Racket Solution:

```
(define exam-room%
(class object%
(super-new)

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

; seat : -> exact-integer?
(define/public (seat)
)
; leave : exact-integer? -> void?
(define/public (leave p)
)))

;; Your exam-room% object will be instantiated and called as such:
;; (define obj (new exam-room% [n n]))
;; (define param_1 (send obj seat))
;; (send obj leave p)
```