Problem 2069: Walking Robot Simulation II

Problem Information

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

Problem Description

A

width x height

grid is on an XY-plane with the

bottom-left

cell at

(0, 0)

and the

top-right

cell at

(width - 1, height - 1)

. The grid is aligned with the four cardinal directions (

"North"

,

"East"

,

"South"

, and

"West"

). A robot is

initially

at cell

(0, 0)

facing direction

"East"

.

The robot can be instructed to move for a specific number of

steps

. For each step, it does the following.

Attempts to move

forward one

cell in the direction it is facing.

If the cell the robot is

moving to

is

out of bounds

, the robot instead

turns

90 degrees

counterclockwise

and retries the step.

After the robot finishes moving the number of steps required, it stops and awaits the next instruction.

Implement the

Robot

class:

Robot(int width, int height)

Initializes the

width x height

grid with the robot at

(0, 0)

facing

"East"

.

void step(int num)

Instructs the robot to move forward

num

steps.

int[] getPos()

Returns the current cell the robot is at, as an array of length 2,

[x, y]

.

String getDir()

Returns the current direction of the robot,

"North"

,

"East"

,

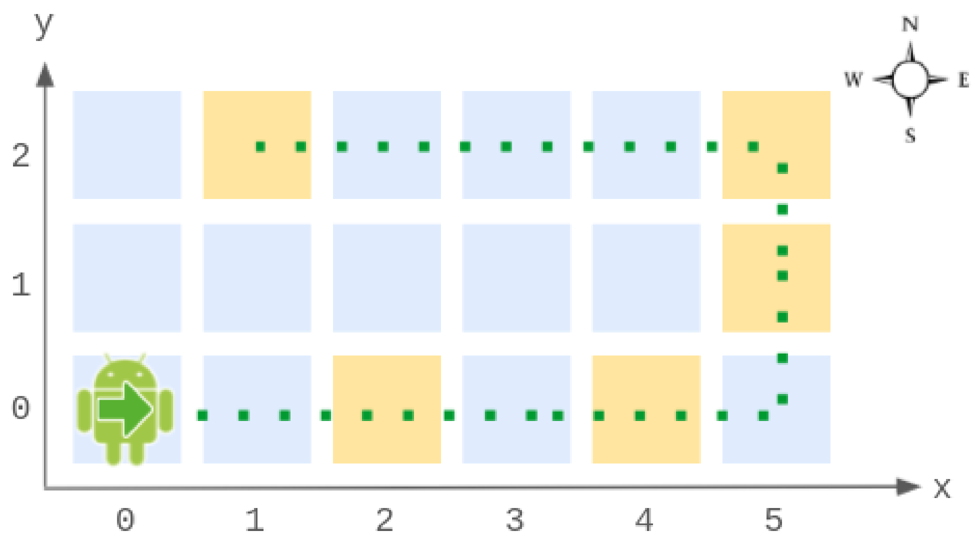
"South"

, or

"West"

.

Example 1:



Input

["Robot", "step", "step", "getPos", "getDir", "step", "step", "step", "getPos", "getDir"] [[6, 3], [2], [2], [], [], [2], [1], [4], [], []]

Output

[null, null, null, [4, 0], "East", null, null, null, [1, 2], "West"]

Explanation

Robot robot = new Robot(6, 3); // Initialize the grid and the robot at (0, 0) facing East. robot.step(2); // It moves two steps East to (2, 0), and faces East. robot.step(2); // It moves two steps East to (4, 0), and faces East. robot.getPos(); // return [4, 0] robot.getDir(); // return "East" robot.step(2); // It moves one step East to (5, 0), and faces East. // Moving the next step East would be out of bounds, so it turns and faces North. // Then, it moves one step North to (5, 1), and faces North. robot.step(1); // It moves one step North to (5, 2), and faces

North

(not West). robot.step(4); // Moving the next step North would be out of bounds, so it turns and faces West. // Then, it moves four steps West to (1, 2), and faces West. robot.getPos(); // return [1, 2] robot.getDir(); // return "West"

Constraints:

2 <= width, height <= 100

1 <= num <= 10

5

At most

10

4

calls

in total

will be made to

step

,

getPos

, and

getDir

.

Code Snippets

C++:

```
class Robot {
public:
Robot(int width, int height) {

}

void step(int num) {
}

vector<int> getPos() {

}

string getDir() {

}
};

/**
* Your Robot object will be instantiated and called as such:
* Robot* obj = new Robot(width, height);
* obj->step(num);
* vector<int> param_2 = obj->getPos();
* string param_3 = obj->getDir();
*/
```

Java:

```
class Robot {

public Robot(int width, int height) {

}

public void step(int num) {

}

public int[] getPos() {

}

public String getDir() {

}
}

/**
```

```
* Your Robot object will be instantiated and called as such:
* Robot obj = new Robot(width, height);
* obj.step(num);
* int[] param_2 = obj.getPos();
* String param_3 = obj.getDir();
*/
```

Python3:

```
class Robot:

def __init__(self, width: int, height: int):



def step(self, num: int) -> None:



def getPos(self) -> List[int]:



def getDir(self) -> str:




# Your Robot object will be instantiated and called as such:
# obj = Robot(width, height)
# obj.step(num)
# param_2 = obj.getPos()
# param_3 = obj.getDir()
```

Python:

```
class Robot(object):

def __init__(self, width, height):
"""
:type width: int
:type height: int
"""



def step(self, num):
```

```
"""
:type num: int
:rtype: None
"""



def getPos(self):
"""
:rtype: List[int]
"""



def getDir(self):
"""
:rtype: str
"""




# Your Robot object will be instantiated and called as such:
# obj = Robot(width, height)
# obj.step(num)
# param_2 = obj.getPos()
# param_3 = obj.getDir()
```

JavaScript:

```
/**
* @param {number} width
* @param {number} height
*/
var Robot = function(width, height) {

};

/**
* @param {number} num
* @return {void}
*/
Robot.prototype.step = function(num) {

};
```

```
/**
* @return {number[]}
*/
Robot.prototype.getPos = function() {

};

/**
* @return {string}
*/
Robot.prototype.getDir = function() {

};

/**
* Your Robot object will be instantiated and called as such:
* var obj = new Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
*/
```

TypeScript:

```
class Robot {
constructor(width: number, height: number) {

}

step(num: number): void {

}

getPos(): number[] {

}

getDir(): string {

}
}
```

```
/**
* Your Robot object will be instantiated and called as such:
* var obj = new Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
*/
```

C#:

```
public class Robot {

public Robot(int width, int height) {

}

public void Step(int num) {

}

public int[] GetPos() {

}

public string GetDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* Robot obj = new Robot(width, height);
* obj.Step(num);
* int[] param_2 = obj.GetPos();
* string param_3 = obj.GetDir();
*/
```

C:

```
typedef struct {

} Robot;


Robot* robotCreate(int width, int height) {

}

void robotStep(Robot* obj, int num) {

}

int* robotGetPos(Robot* obj, int* retSize) {

}

char* robotGetDir(Robot* obj) {

}

void robotFree(Robot* obj) {

}

/**
* Your Robot struct will be instantiated and called as such:
* Robot* obj = robotCreate(width, height);
* robotStep(obj, num);

* int* param_2 = robotGetPos(obj, retSize);

* char* param_3 = robotGetDir(obj);

* robotFree(obj);
*/
```

Go:

```
type Robot struct {
}


func Constructor(width int, height int) Robot {

}


func (this *Robot) Step(num int) {

}


func (this *Robot) GetPos() []int {

}


func (this *Robot) GetDir() string {

}


/**
* Your Robot object will be instantiated and called as such:
* obj := Constructor(width, height);
* obj.Step(num);
* param_2 := obj.GetPos();
* param_3 := obj.GetDir();
*/
```

Kotlin:

```
class Robot(width: Int, height: Int) {

fun step(num: Int) {

}

fun getPos(): IntArray {

}
fun getDir(): String {

}

}

/**
* Your Robot object will be instantiated and called as such:
* var obj = Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
*/
```

Swift:

```
class Robot {

init(_ width: Int, _ height: Int) {

}

func step(_ num: Int) {

}

func getPos() -> [Int] {

}

func getDir() -> String {

}
}

/**
* Your Robot object will be instantiated and called as such:
* let obj = Robot(width, height)
* obj.step(num)
* let ret_2: [Int] = obj.getPos()
* let ret_3: String = obj.getDir()
*/
```

Rust:

```
struct Robot {

}



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

fn new(width: i32, height: i32) -> Self {

}

fn step(&self, num: i32) {

}

fn get_pos(&self) -> Vec<i32> {

}

fn get_dir(&self) -> String {

}
}

/**
* Your Robot object will be instantiated and called as such:
* let obj = Robot::new(width, height);
* obj.step(num);
* let ret_2: Vec<i32> = obj.get_pos();
* let ret_3: String = obj.get_dir();
*/
```

Ruby:

```
class Robot

=begin
:type width: Integer
:type height: Integer
=end
def initialize(width, height)

end



=begin
:type num: Integer
:rtype: Void
=end
def step(num)

end



=begin
:rtype: Integer[]
=end
def get_pos()

end



=begin
:rtype: String
=end
def get_dir()

end



end

# Your Robot object will be instantiated and called as such:
# obj = Robot.new(width, height)
# obj.step(num)
# param_2 = obj.get_pos()
# param_3 = obj.get_dir()
```

PHP:

```
class Robot {
/**
* @param Integer $width
* @param Integer $height
*/
function __construct($width, $height) {

}

/**
* @param Integer $num
* @return NULL
*/
function step($num) {

}

/**
* @return Integer[]
*/
function getPos() {

}

/**
* @return String
*/
function getDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* $obj = Robot($width, $height);
* $obj->step($num);
* $ret_2 = $obj->getPos();
* $ret_3 = $obj->getDir();
*/
```

Dart:

```
class Robot {

Robot(int width, int height) {

}

void step(int num) {

}

List<int> getPos() {

}

String getDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* Robot obj = Robot(width, height);
* obj.step(num);
* List<int> param2 = obj.getPos();
* String param3 = obj.getDir();
*/
```

Scala:

```
class Robot(_width: Int, _height: Int) {

def step(num: Int): Unit = {

}

def getPos(): Array[Int] = {
}

def getDir(): String = {

}

}

/**
* Your Robot object will be instantiated and called as such:
* val obj = new Robot(width, height)
* obj.step(num)
* val param_2 = obj.getPos()
* val param_3 = obj.getDir()
*/
```

Elixir:

```
defmodule Robot do
@spec init_(width :: integer, height :: integer) :: any
def init_(width, height) do

end

@spec step(num :: integer) :: any
def step(num) do

end

@spec get_pos() :: [integer]
def get_pos() do

end

@spec get_dir() :: String.t
def get_dir() do

end
end

# Your functions will be called as such:
# Robot.init_(width, height)
# Robot.step(num)
# param_2 = Robot.get_pos()
# param_3 = Robot.get_dir()

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

Erlang:

```
-spec robot_init_(Width :: integer(), Height :: integer()) -> any().
robot_init_(Width, Height) ->
.

-spec robot_step(Num :: integer()) -> any().
robot_step(Num) ->
.

-spec robot_get_pos() -> [integer()].
robot_get_pos() ->
.

-spec robot_get_dir() -> unicode:unicode_binary().
robot_get_dir() ->
.



%% Your functions will be called as such:
%% robot_init_(Width, Height),
%% robot_step(Num),
%% Param_2 = robot_get_pos(),
%% Param_3 = robot_get_dir(),

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

Racket:

```
(define robot%
(class object%
(super-new)

; width : exact-integer?
; height : exact-integer?
(init-field
width
height)

; step : exact-integer? -> void?
(define/public (step num)
)
; get-pos : -> (listof exact-integer?)
(define/public (get-pos)
)
; get-dir : -> string?
(define/public (get-dir)
)))

;; Your robot% object will be instantiated and called as such:
;; (define obj (new robot% [width width] [height height]))
;; (send obj step num)
;; (define param_2 (send obj get-pos))
;; (define param_3 (send obj get-dir))
```

Solutions

C++ Solution:

```
/*
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Robot {
public:
Robot(int width, int height) {

}
void step(int num) {

}

vector<int> getPos() {

}

string getDir() {

}
};

/**
* Your Robot object will be instantiated and called as such:
* Robot* obj = new Robot(width, height);
* obj->step(num);
* vector<int> param_2 = obj->getPos();
* string param_3 = obj->getDir();
*/
```

Java Solution:

```
/**
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Robot {

public Robot(int width, int height) {

}

public void step(int num) {
}

public int[] getPos() {

}

public String getDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* Robot obj = new Robot(width, height);
* obj.step(num);
* int[] param_2 = obj.getPos();
* String param_3 = obj.getDir();
*/
```

Python3 Solution:

```
"""
Problem: Walking Robot Simulation II
Difficulty: Medium
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Robot:

def __init__(self, width: int, height: int):



def step(self, num: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Robot(object):

def __init__(self, width, height):
"""
:type width: int
:type height: int
"""



def step(self, num):
"""
:type num: int
:rtype: None
"""



def getPos(self):
"""
:rtype: List[int]
"""



def getDir(self):
"""
:rtype: str
"""




# Your Robot object will be instantiated and called as such:
# obj = Robot(width, height)
# obj.step(num)
# param_2 = obj.getPos()
# param_3 = obj.getDir()
```

JavaScript Solution:

```
/**
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} width
* @param {number} height
*/
var Robot = function(width, height) {

};

/**
* @param {number} num
* @return {void}
*/
Robot.prototype.step = function(num) {

};

/**
* @return {number[]}
*/
Robot.prototype.getPos = function() {

};

/**
* @return {string}
*/
Robot.prototype.getDir = function() {

};

/**
* Your Robot object will be instantiated and called as such:
* var obj = new Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
```

```
*/
```

TypeScript Solution:

```
/**
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Robot {
constructor(width: number, height: number) {

}

step(num: number): void {

}

getPos(): number[] {

}

getDir(): string {

}
}

/**
* Your Robot object will be instantiated and called as such:
* var obj = new Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
*/
```

C# Solution:

```
/*
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Robot {

public Robot(int width, int height) {

}

public void Step(int num) {

}

public int[] GetPos() {

}

public string GetDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* Robot obj = new Robot(width, height);
* obj.Step(num);
* int[] param_2 = obj.GetPos();
* string param_3 = obj.GetDir();
*/
```

C Solution:

```
/*
* Problem: Walking Robot Simulation II
* Difficulty: Medium
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/




typedef struct {

} Robot;


Robot* robotCreate(int width, int height) {

}

void robotStep(Robot* obj, int num) {

}

int* robotGetPos(Robot* obj, int* retSize) {

}

char* robotGetDir(Robot* obj) {

}

void robotFree(Robot* obj) {

}

/**
* Your Robot struct will be instantiated and called as such:
* Robot* obj = robotCreate(width, height);
* robotStep(obj, num);

* int* param_2 = robotGetPos(obj, retSize);
* char* param_3 = robotGetDir(obj);

* robotFree(obj);
*/
```

Go Solution:

```
// Problem: Walking Robot Simulation II
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

type Robot struct {

}


func Constructor(width int, height int) Robot {

}


func (this *Robot) Step(num int) {

}


func (this *Robot) GetPos() []int {

}


func (this *Robot) GetDir() string {

}


/**
```

```
* Your Robot object will be instantiated and called as such:
* obj := Constructor(width, height);
* obj.Step(num);
* param_2 := obj.GetPos();
* param_3 := obj.GetDir();
*/
```

Kotlin Solution:

```
class Robot(width: Int, height: Int) {

fun step(num: Int) {

}

fun getPos(): IntArray {

}

fun getDir(): String {

}

}

/**
* Your Robot object will be instantiated and called as such:
* var obj = Robot(width, height)
* obj.step(num)
* var param_2 = obj.getPos()
* var param_3 = obj.getDir()
*/
```

Swift Solution:

```
class Robot {

init(_ width: Int, _ height: Int) {

}


func step(_ num: Int) {

}

func getPos() -> [Int] {

}

func getDir() -> String {

}
}

/**
* Your Robot object will be instantiated and called as such:
* let obj = Robot(width, height)
* obj.step(num)
* let ret_2: [Int] = obj.getPos()
* let ret_3: String = obj.getDir()
*/
```

Rust Solution:

```
// Problem: Walking Robot Simulation II
// Difficulty: Medium
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

struct Robot {

}



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

```
impl Robot {

fn new(width: i32, height: i32) -> Self {

}

fn step(&self, num: i32) {

}

fn get_pos(&self) -> Vec<i32> {

}

fn get_dir(&self) -> String {

}
}

/**
* Your Robot object will be instantiated and called as such:
* let obj = Robot::new(width, height);
* obj.step(num);
* let ret_2: Vec<i32> = obj.get_pos();
* let ret_3: String = obj.get_dir();
*/
```

Ruby Solution:

```
class Robot

=begin
:type width: Integer
:type height: Integer
=end
def initialize(width, height)

end



=begin
```

```
:type num: Integer
:rtype: Void
=end
def step(num)

end



=begin
:rtype: Integer[]
=end
def get_pos()

end



=begin
:rtype: String
=end
def get_dir()

end



end

# Your Robot object will be instantiated and called as such:
# obj = Robot.new(width, height)
# obj.step(num)
# param_2 = obj.get_pos()
# param_3 = obj.get_dir()
```

PHP Solution:

```
class Robot {
/**
* @param Integer $width
* @param Integer $height
*/
function __construct($width, $height) {
}

/**
* @param Integer $num
* @return NULL
*/
function step($num) {

}

/**
* @return Integer[]
*/
function getPos() {

}

/**
* @return String
*/
function getDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* $obj = Robot($width, $height);
* $obj->step($num);
* $ret_2 = $obj->getPos();
* $ret_3 = $obj->getDir();
*/
```

Dart Solution:

```
class Robot {

Robot(int width, int height) {

}
void step(int num) {

}

List<int> getPos() {

}

String getDir() {

}
}

/**
* Your Robot object will be instantiated and called as such:
* Robot obj = Robot(width, height);
* obj.step(num);
* List<int> param2 = obj.getPos();
* String param3 = obj.getDir();
*/
```

Scala Solution:

```
class Robot(_width: Int, _height: Int) {

def step(num: Int): Unit = {

}

def getPos(): Array[Int] = {

}

def getDir(): String = {

}

}

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

```
* val obj = new Robot(width, height)
* obj.step(num)
* val param_2 = obj.getPos()
* val param_3 = obj.getDir()
*/
```

Elixir Solution:

```
defmodule Robot do
@spec init_(width :: integer, height :: integer) :: any
def init_(width, height) do

end

@spec step(num :: integer) :: any
def step(num) do

end

@spec get_pos() :: [integer]
def get_pos() do

end

@spec get_dir() :: String.t
def get_dir() do

end
end

# Your functions will be called as such:
# Robot.init_(width, height)
# Robot.step(num)
# param_2 = Robot.get_pos()
# param_3 = Robot.get_dir()

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

Erlang Solution:

```
-spec robot_init_(Width :: integer(), Height :: integer()) -> any().
robot_init_(Width, Height) ->
.

-spec robot_step(Num :: integer()) -> any().
robot_step(Num) ->
.

-spec robot_get_pos() -> [integer()].
robot_get_pos() ->
.

-spec robot_get_dir() -> unicode:unicode_binary().
robot_get_dir() ->
.


%% Your functions will be called as such:
%% robot_init_(Width, Height),
%% robot_step(Num),
%% Param_2 = robot_get_pos(),
%% Param_3 = robot_get_dir(),

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

Racket Solution:

```
(define robot%
(class object%
(super-new)

; width : exact-integer?
; height : exact-integer?
(init-field
width
height)

; step : exact-integer? -> void?
(define/public (step num)
)
; get-pos : -> (listof exact-integer?)
(define/public (get-pos)
)
; get-dir : -> string?
(define/public (get-dir)
)))

;; Your robot% object will be instantiated and called as such:
;; (define obj (new robot% [width width] [height height]))
;; (send obj step num)
;; (define param_2 (send obj get-pos))
;; (define param_3 (send obj get-dir))
```