Problem 3178: Find the Child Who Has the Ball After K Seconds

Problem Information

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

Problem Description

You are given two

positive

integers

n

and

k

. There are

n

children numbered from

0

to

n - 1

standing in a queue

in order

from left to right.

Initially, child 0 holds a ball and the direction of passing the ball is towards the right direction. After each second, the child holding the ball passes it to the child next to them. Once the ball reaches

either

end of the line, i.e. child 0 or child

n - 1

, the direction of passing is

reversed

.

Return the number of the child who receives the ball after

k

seconds.

Example 1:

Input:

n = 3, k = 5

Output:

1

Explanation:

Time elapsed

Children

0

[

0

, 1, 2]

1

[0,

1

, 2]

2

[0, 1,

2

]

3

[0,

1

, 2]

4

[

0

, 1, 2]

5

[0,

1

, 2]

Example 2:

Input:

n = 5, k = 6

Output:

2

Explanation:

Time elapsed

Children

0

[

0

, 1, 2, 3, 4]

1

[0,

1

, 2, 3, 4]

2

[0, 1,

2

, 3, 4]

3

[0, 1, 2,

3

, 4]

4

[0, 1, 2, 3,

4

]

5

[0, 1, 2,

3

, 4]

6

[0, 1,

2

, 3, 4]

Example 3:

Input:

n = 4, k = 2

Output:

2

Explanation:

Time elapsed

Children

0

[

0

, 1, 2, 3]

1

[0,

1

, 2, 3]

2

[0, 1,

2

, 3]

Constraints:

2 <= n <= 50

1 <= k <= 50

Note:

This question is the same as

2582: Pass the Pillow.

Code Snippets

C++:

```
class Solution {
public:
int numberOfChild(int n, int k) {

}
};
```

Java:

```
class Solution {
public int numberOfChild(int n, int k) {

}
}
```

Python3:

```
class Solution:
def numberOfChild(self, n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfChild(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var numberOfChild = function(n, k) {

};
```

TypeScript:

```
function numberOfChild(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfChild(int n, int k) {

}
}
```

C:

```
int numberOfChild(int n, int k) {

}
```

Go:

```
func numberOfChild(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfChild(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfChild(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_child(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def number_of_child(n, k)

end
```

PHP:

```
class Solution {
```

```
/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function numberOfChild($n, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfChild(int n, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfChild(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_child(n :: integer, k :: integer) :: integer
def number_of_child(n, k) do

end
end
```

Erlang:

```
-spec number_of_child(N :: integer(), K :: integer()) -> integer().
number_of_child(N, K) ->
.
```

Racket:

```
(define/contract (number-of-child n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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 Solution {
public:
int numberOfChild(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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 Solution {
public int numberOfChild(int n, int k) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Find the Child Who Has the Ball After K Seconds
Difficulty: Easy
Tags: math, queue

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 Solution:
def numberOfChild(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfChild(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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
* @param {number} k
* @return {number}
*/
var numberOfChild = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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
*/

function numberOfChild(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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 Solution {
public int NumberOfChild(int n, int k) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Find the Child Who Has the Ball After K Seconds
* Difficulty: Easy
* Tags: math, queue
*
* 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
*/

int numberOfChild(int n, int k) {

}
```

Go Solution:

```
// Problem: Find the Child Who Has the Ball After K Seconds
// Difficulty: Easy
// Tags: math, queue
//
// 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

func numberOfChild(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfChild(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfChild(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Find the Child Who Has the Ball After K Seconds
// Difficulty: Easy
// Tags: math, queue
//
// 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

impl Solution {
pub fn number_of_child(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def number_of_child(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function numberOfChild($n, $k) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int numberOfChild(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfChild(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_child(n :: integer, k :: integer) :: integer
def number_of_child(n, k) do

end
end
```

Erlang Solution:

```
-spec number_of_child(N :: integer(), K :: integer()) -> integer().
number_of_child(N, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-child n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```