Problem 70: Climbing Stairs

Problem Information

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

Problem Description

You are climbing a staircase. It takes

n

steps to reach the top.

Each time you can either climb

1

or

2

steps. In how many distinct ways can you climb to the top?

Example 1:

Input:

n = 2

Output:

2

Explanation:

There are two ways to climb to the top. 1. 1 step + 1 step 2. 2 steps

Example 2:

Input:

n = 3

Output:

3

Explanation:

There are three ways to climb to the top. 1. 1 step + 1 step + 1 step 2. 1 step + 2 steps 3. 2 steps + 1 step

Constraints:

1 <= n <= 45

Code Snippets

C++:

```
class Solution {
public:
int climbStairs(int n) {

}
};
```

Java:

```
class Solution {
public int climbStairs(int n) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function climbStairs(n: number): number {

};
```

C#:

```
public class Solution {
public int ClimbStairs(int n) {

}
}
```

C:

```
int climbStairs(int n) {

}
```

Go:

```
func climbStairs(n int) int {

}
```

Kotlin:

```
class Solution {
fun climbStairs(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func climbStairs(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn climb_stairs(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def climb_stairs(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function climbStairs($n) {

}
}
```

Dart:

```
class Solution {
int climbStairs(int n) {

}
}
```

Scala:

```
object Solution {
def climbStairs(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec climb_stairs(n :: integer) :: integer
def climb_stairs(n) do

end
end
```

Erlang:

```
-spec climb_stairs(N :: integer()) -> integer().
climb_stairs(N) ->
.
```

Racket:

```
(define/contract (climb-stairs n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int climbStairs(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int climbStairs(int n) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Climbing Stairs
Difficulty: Easy
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function climbStairs(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int ClimbStairs(int n) {

}
}
```

C Solution:

```
/*
* Problem: Climbing Stairs
* Difficulty: Easy
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int climbStairs(int n) {

}
```

Go Solution:

```
// Problem: Climbing Stairs
// Difficulty: Easy
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func climbStairs(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun climbStairs(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func climbStairs(_ n: Int) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Climbing Stairs
// Difficulty: Easy
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn climb_stairs(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def climb_stairs(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function climbStairs($n) {

}
}
```

Dart Solution:

```
class Solution {
int climbStairs(int n) {

}
}
```

Scala Solution:

```
object Solution {
def climbStairs(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec climb_stairs(n :: integer) :: integer
def climb_stairs(n) do

end
end
```

Erlang Solution:

```
-spec climb_stairs(N :: integer()) -> integer().
climb_stairs(N) ->
.
```

Racket Solution:

```
(define/contract (climb-stairs n)
(-> exact-integer? exact-integer?)
)
```