Problem 9: Palindrome Number

Problem Information

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

Problem Description

Given an integer

x

, return

true

if

x

is a

palindrome

, and

false

otherwise

.

Example 1:

Input:

x = 121

Output:

true

Explanation:

121 reads as 121 from left to right and from right to left.

Example 2:

Input:

x = -121

Output:

false

Explanation:

From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3:

Input:

x = 10

Output:

false

Explanation:

Reads 01 from right to left. Therefore it is not a palindrome.

Constraints:

-2

31

<= x <= 2

31

- 1

Follow up:

Could you solve it without converting the integer to a string?

Code Snippets

C++:

```
class Solution {
public:
bool isPalindrome(int x) {

}
};
```

Java:

```
class Solution {
public boolean isPalindrome(int x) {

}
}
```

Python3:

```
class Solution:
def isPalindrome(self, x: int) -> bool:
```

Python:

```
class Solution(object):
def isPalindrome(self, x):
"""
:type x: int
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number} x
* @return {boolean}
*/
var isPalindrome = function(x) {

};
```

TypeScript:

```
function isPalindrome(x: number): boolean {

};
```

C#:

```
public class Solution {
public bool IsPalindrome(int x) {

}
}
```

C:

```
bool isPalindrome(int x) {

}
```

Go:

```
func isPalindrome(x int) bool {

}
```

Kotlin:

```
class Solution {
fun isPalindrome(x: Int): Boolean {

}
}
```

Swift:

```
class Solution {
func isPalindrome(_ x: Int) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_palindrome(x: i32) -> bool {

}
}
```

Ruby:

```
# @param {Integer} x
# @return {Boolean}
def is_palindrome(x)

end
```

PHP:

```
class Solution {

/**
* @param Integer $x
* @return Boolean
```

```
*/
function isPalindrome($x) {

}
}
```

Dart:

```
class Solution {
bool isPalindrome(int x) {

}
}
```

Scala:

```
object Solution {
def isPalindrome(x: Int): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_palindrome(x :: integer) :: boolean
def is_palindrome(x) do

end
end
```

Erlang:

```
-spec is_palindrome(X :: integer()) -> boolean().
is_palindrome(X) ->
.
```

Racket:

```
(define/contract (is-palindrome x)
(-> exact-integer? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Palindrome Number
* Difficulty: Easy
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool isPalindrome(int x) {

}
};
```

Java Solution:

```
/**
* Problem: Palindrome Number
* Difficulty: Easy
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean isPalindrome(int x) {

}
}
```

Python3 Solution:

```
"""
Problem: Palindrome Number
Difficulty: Easy
Tags: string, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def isPalindrome(self, x: int) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isPalindrome(self, x):
"""
:type x: int
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Palindrome Number
* Difficulty: Easy
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} x
* @return {boolean}
*/
var isPalindrome = function(x) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Palindrome Number
* Difficulty: Easy
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function isPalindrome(x: number): boolean {

};
```

C# Solution:

```
/*
* Problem: Palindrome Number
* Difficulty: Easy
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool IsPalindrome(int x) {

}
}
```

C Solution:

```
/*
* Problem: Palindrome Number
* Difficulty: Easy
```

```
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool isPalindrome(int x) {

}
```

Go Solution:

```
// Problem: Palindrome Number
// Difficulty: Easy
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func isPalindrome(x int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPalindrome(x: Int): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isPalindrome(_ x: Int) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Palindrome Number
// Difficulty: Easy
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn is_palindrome(x: i32) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer} x
# @return {Boolean}
def is_palindrome(x)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $x
* @return Boolean
*/
function isPalindrome($x) {

}
}
```

Dart Solution:

```
class Solution {
bool isPalindrome(int x) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def isPalindrome(x: Int): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_palindrome(x :: integer) :: boolean
def is_palindrome(x) do

end
end
```

Erlang Solution:

```
-spec is_palindrome(X :: integer()) -> boolean().
is_palindrome(X) ->
.
```

Racket Solution:

```
(define/contract (is-palindrome x)
(-> exact-integer? boolean?)
)
```