Problem 284: Peeking Iterator

Problem Information

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

Problem Description

Design an iterator that supports the

peek

operation on an existing iterator in addition to the

hasNext

and the

next

operations.

Implement the

PeekingIterator

class:

PeekingIterator(Iterator<int> nums)

Initializes the object with the given integer iterator

iterator

.

int next()

Returns the next element in the array and moves the pointer to the next element.

boolean hasNext()

Returns

true

if there are still elements in the array.

int peek()

Returns the next element in the array

without

moving the pointer.

Note:

Each language may have a different implementation of the constructor and

Iterator

, but they all support the

int next()

and

boolean hasNext()

functions.

Example 1:

Input

["PeekingIterator", "next", "peek", "next", "next", "hasNext"] [[[1, 2, 3]], [], [], [], [], []]

Output

[null, 1, 2, 2, 3, false]

Explanation

PeekingIterator peekingIterator = new PeekingIterator([1, 2, 3]); // [

1

,2,3] peekingIterator.next(); // return 1, the pointer moves to the next element [1,

2

,3]. peekingIterator.peek(); // return 2, the pointer does not move [1,

2

,3]. peekingIterator.next(); // return 2, the pointer moves to the next element [1,2,

3

] peekingIterator.next(); // return 3, the pointer moves to the next element [1,2,3]
peekingIterator.hasNext(); // return False

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 1000

All the calls to

next

and

peek

are valid.

At most

1000

calls will be made to

next

,

hasNext

, and

peek

.

Follow up:

How would you extend your design to be generic and work with all types, not just integer?

Code Snippets

C++:

```
/*
* Below is the interface for Iterator, which is already defined for you.
* **DO NOT** modify the interface for Iterator.
*
* class Iterator {
* struct Data;
* Data* data;
* public:
* Iterator(const vector<int>& nums);
* Iterator(const Iterator& iter);
*
* // Returns the next element in the iteration.
* int next();
*
* // Returns true if the iteration has more elements.
* bool hasNext() const;
* };
*/

class PeekingIterator : public Iterator {
public:
PeekingIterator(const vector<int>& nums) : Iterator(nums) {
// Initialize any member here.
// **DO NOT** save a copy of nums and manipulate it directly.
// You should only use the Iterator interface methods.

}

// Returns the next element in the iteration without advancing the iterator.
int peek() {

}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
int next() {

}

bool hasNext() const {

}
};
```

Java:

```
// Java Iterator interface reference:
// https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html
```

```
class PeekingIterator implements Iterator<Integer> {
public PeekingIterator(Iterator<Integer> iterator) {
// initialize any member here.

}

// Returns the next element in the iteration without advancing the iterator.
public Integer peek() {

}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
@Override
public Integer next() {

}

@Override
public boolean hasNext() {

}
}
```

Python3:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator:
# def __init__(self, nums):
# """
# Initializes an iterator object to the beginning of a list.
# :type nums: List[int]
# """
#
# def hasNext(self):
# """
# Returns true if the iteration has more elements.
# :rtype: bool
# """
#
```

```
# def next(self):
# """
# Returns the next element in the iteration.
# :rtype: int
# """

class PeekingIterator:
def __init__(self, iterator):
"""
Initialize your data structure here.
:type iterator: Iterator
"""



def peek(self):
"""
Returns the next element in the iteration without advancing the iterator.
:rtype: int
"""



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



def hasNext(self):
"""
:rtype: bool
"""



# Your PeekingIterator object will be instantiated and called as such:
# iter = PeekingIterator(Iterator(nums))
# while iter.hasNext():
# val = iter.peek() # Get the next element but not advance the iterator.
# iter.next() # Should return the same value as [val].
```

Python:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator(object):
# def __init__(self, nums):
# """
# Initializes an iterator object to the beginning of a list.
# :type nums: List[int]
# """
#
# def hasNext(self):
# """
# Returns true if the iteration has more elements.
# :rtype: bool
# """
#
# def next(self):
# """
# Returns the next element in the iteration.
# :rtype: int
# """

class PeekingIterator(object):
def __init__(self, iterator):
"""
Initialize your data structure here.
:type iterator: Iterator
"""



def peek(self):
"""
Returns the next element in the iteration without advancing the iterator.
:rtype: int
"""



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



def hasNext(self):
```

```
"""
:rtype: bool
"""



# Your PeekingIterator object will be instantiated and called as such:
# iter = PeekingIterator(Iterator(nums))
# while iter.hasNext():
# val = iter.peek() # Get the next element but not advance the iterator.
# iter.next() # Should return the same value as [val].
```

JavaScript:

```
/**
* // This is the Iterator's API interface.
* // You should not implement it, or speculate about its implementation.
* function Iterator() {
* @ return {number}
* this.next = function() { // return the next number of the iterator
* ...
* };
*
* @return {boolean}
* this.hasNext = function() { // return true if it still has numbers
* ...
* };
* };
*/

/**
* @param {Iterator} iterator
*/
var PeekingIterator = function(iterator) {

};

/**
* @return {number}
*/
PeekingIterator.prototype.peek = function() {

};
```

```
/**
* @return {number}
*/
PeekingIterator.prototype.next = function() {

};

/**
* @return {boolean}
*/
PeekingIterator.prototype.hasNext = function() {

};

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(arr)
* var param_1 = obj.peek()
* var param_2 = obj.next()
* var param_3 = obj.hasNext()
*/
```

TypeScript:

```
/**
* // This is the Iterator's API interface.
* // You should not implement it, or speculate about its implementation
* class Iterator {
* hasNext(): boolean {}
*
* next(): number {}
* }
*/

class PeekingIterator {
constructor(iterator: Iterator) {

}

peek(): number {
}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(iterator)
* var param_1 = obj.peek()
* var param_2 = obj.next()
* var param_3 = obj.hasNext()
*/
```

C#:

```
// C# IEnumerator interface reference:
// https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerator
?view=netframework-4.8

class PeekingIterator {
// iterators refers to the first element of the array.
public PeekingIterator(IEnumerator<int> iterator) {
// initialize any member here.
}

// Returns the next element in the iteration without advancing the iterator.
public int Peek() {

}

// Returns the next element in the iteration and advances the iterator.
public int Next() {

}

// Returns false if the iterator is refering to the end of the array of true
otherwise.
public bool HasNext() {

}
}
```

C:

```
/*
* struct Iterator {
* // Returns true if the iteration has more elements.
* bool (*hasNext)();
*
* // Returns the next element in the iteration.
* int (*next)();
* };
*/

struct PeekingIterator {

};

struct PeekingIterator* Constructor(struct Iterator* iter) {
struct PeekingIterator* piter = malloc(sizeof(struct PeekingIterator));
piter->iterator = iter;
piter->hasPeeked = false;
return piter;
}

int peek(struct PeekingIterator* obj) {

}

int next(struct PeekingIterator* obj) {

}

bool hasNext(struct PeekingIterator* obj) {

}

/**
* Your PeekingIterator struct will be instantiated and called as such:
```

```
* PeekingIterator* obj = peekingIteratorCreate(arr, arrSize);
* int param_1 = peek(obj);
* int param_2 = next(obj);
* bool param_3 = hasNext(obj);
* peekingIteratorFree(obj);
*/
```

Go:

```
/* Below is the interface for Iterator, which is already defined for you.
*
* type Iterator struct {
*
* }
*
* func (this *Iterator) hasNext() bool {
* // Returns true if the iteration has more elements.
* }
*
* func (this *Iterator) next() int {
* // Returns the next element in the iteration.
* }
*/

type PeekingIterator struct {

}

func Constructor(iter *Iterator) *PeekingIterator {

}

func (this *PeekingIterator) hasNext() bool {

}

func (this *PeekingIterator) next() int {

}

func (this *PeekingIterator) peek() int {
}
```

Kotlin:

```
// Kotlin Iterator reference:
// https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-iterator/

class PeekingIterator(iterator:Iterator<Int>):Iterator<Int> {
fun peek(): Int {

}

override fun next(): Int {

}

override fun hasNext(): Boolean {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = PeekingIterator(arr)
* var param_1 = obj.next()
* var param_2 = obj.peek()
* var param_3 = obj.hasNext()
*/
```

Swift:

```
// Swift IndexingIterator refernence:
// https://developer.apple.com/documentation/swift/indexingiterator

class PeekingIterator {
init(_ arr: IndexingIterator<Array<Int>>) {

}

func next() -> Int {

}
func peek() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* let obj = PeekingIterator(arr)
* let ret_1: Int = obj.next()
* let ret_2: Int = obj.peek()
* let ret_3: Bool = obj.hasNext()
*/
```

Ruby:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator
# def initialize(v)
#
# end
#
# def hasNext()
# Returns true if the iteration has more elements.
# end
#
# def next()
# Returns the next element in the iteration.
# end
# end

class PeekingIterator
# @param {Iterator} iter
def initialize(iter)

end
```

```

# Returns true if the iteration has more elements.
# @return {boolean}
def hasNext()

end

# Returns the next element in the iteration.
# @return {integer}
def next()

end

# Returns the next element in the iteration without advancing the iterator.
# @return {integer}
def peek()

end
end
```

PHP:

```
// PHP ArrayIterator reference:
// https://www.php.net/arrayiterator

class PeekingIterator {
/**
* @param ArrayIterator $arr
*/
function __construct($arr) {

}

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

}

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

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* $obj = PeekingIterator($arr);
* $ret_1 = $obj->next();
* $ret_2 = $obj->peek();
* $ret_3 = $obj->hasNext();
*/
```

Scala:

```
// Scala Iterator reference:
// https://www.scala-lang.org/api/2.12.2/scala/collection/Iterator.html

class PeekingIterator(_iterator: Iterator[Int]) {
def peek(): Int = {

}

def next(): Int = {

}

def hasNext(): Boolean = {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(arr)
* var param_1 = obj.next()
* var param_2 = obj.peek()
* var param_3 = obj.hasNext()
*/
```

Solutions

C++ Solution:

```
/*
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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
*/

/*
* Below is the interface for Iterator, which is already defined for you.
* **DO NOT** modify the interface for Iterator.
*
* class Iterator {
* struct Data;
* Data* data;
* public:
* Iterator(const vector<int>& nums);
* Iterator(const Iterator& iter);
*
* // Returns the next element in the iteration.
* int next();
*
* // Returns true if the iteration has more elements.
* bool hasNext() const;
* };
*/

class PeekingIterator : public Iterator {
public:
```

```
PeekingIterator(const vector<int>& nums) : Iterator(nums) {
// Initialize any member here.
// **DO NOT** save a copy of nums and manipulate it directly.
// You should only use the Iterator interface methods.

}

// Returns the next element in the iteration without advancing the iterator.
int peek() {

}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
int next() {

}

bool hasNext() const {

}
};
```

Java Solution:

```
/**
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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
*/

// Java Iterator interface reference:
// https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

class PeekingIterator implements Iterator<Integer> {
public PeekingIterator(Iterator<Integer> iterator) {
// initialize any member here.


}

// Returns the next element in the iteration without advancing the iterator.
public Integer peek() {

}

// hasNext() and next() should behave the same as in the Iterator interface.
// Override them if needed.
@Override
public Integer next() {

}

@Override
public boolean hasNext() {

}
}
```

Python3 Solution:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator:
# def __init__(self, nums):
# """
# Initializes an iterator object to the beginning of a list.
# :type nums: List[int]
# """
#
# def hasNext(self):
# """
# Returns true if the iteration has more elements.
# :rtype: bool
# """
#
# def next(self):
# """
# Returns the next element in the iteration.
# :rtype: int
# """

class PeekingIterator:
def __init__(self, iterator):
"""
Initialize your data structure here.
:type iterator: Iterator
"""



def peek(self):
"""
Returns the next element in the iteration without advancing the iterator.
:rtype: int
"""



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



def hasNext(self):
"""
:rtype: bool
"""



# Your PeekingIterator object will be instantiated and called as such:
# iter = PeekingIterator(Iterator(nums))
# while iter.hasNext():
# val = iter.peek() # Get the next element but not advance the iterator.
# iter.next() # Should return the same value as [val].
```

Python Solution:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator(object):
```

```
# def __init__(self, nums):
# """
# Initializes an iterator object to the beginning of a list.
# :type nums: List[int]
# """
#
# def hasNext(self):
# """
# Returns true if the iteration has more elements.
# :rtype: bool
# """
#
# def next(self):
# """
# Returns the next element in the iteration.
# :rtype: int
# """

class PeekingIterator(object):
def __init__(self, iterator):
"""
Initialize your data structure here.
:type iterator: Iterator
"""



def peek(self):
"""
Returns the next element in the iteration without advancing the iterator.
:rtype: int
"""



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



def hasNext(self):
"""
:rtype: bool
```

```
"""


# Your PeekingIterator object will be instantiated and called as such:
# iter = PeekingIterator(Iterator(nums))
# while iter.hasNext():
# val = iter.peek() # Get the next element but not advance the iterator.
# iter.next() # Should return the same value as [val].
```

JavaScript Solution:

```
/**
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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
*/

/**
* // This is the Iterator's API interface.
* // You should not implement it, or speculate about its implementation.
* function Iterator() {
* @ return {number}
* this.next = function() { // return the next number of the iterator
* ...
* };
*
* @return {boolean}
* this.hasNext = function() { // return true if it still has numbers
* ...
* };
* };
*/

/**
* @param {Iterator} iterator
*/
var PeekingIterator = function(iterator) {
};

/**
* @return {number}
*/
PeekingIterator.prototype.peek = function() {

};

/**
* @return {number}
*/
PeekingIterator.prototype.next = function() {

};

/**
* @return {boolean}
*/
PeekingIterator.prototype.hasNext = function() {

};

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(arr)
* var param_1 = obj.peek()
* var param_2 = obj.next()
* var param_3 = obj.hasNext()
*/
```

TypeScript Solution:

```
/**
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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
*/

/**
* // This is the Iterator's API interface.
* // You should not implement it, or speculate about its implementation
* class Iterator {
* hasNext(): boolean {}
*
* next(): number {}
* }
*/

class PeekingIterator {
constructor(iterator: Iterator) {

}

peek(): number {

}

next(): number {

}

hasNext(): boolean {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(iterator)
* var param_1 = obj.peek()
* var param_2 = obj.next()
* var param_3 = obj.hasNext()
*/
```

C# Solution:

```
/*
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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
*/

// C# IEnumerator interface reference:
// https://docs.microsoft.com/en-us/dotnet/api/system.collections.ienumerator
?view=netframework-4.8

class PeekingIterator {
// iterators refers to the first element of the array.
public PeekingIterator(IEnumerator<int> iterator) {
// initialize any member here.
}

// Returns the next element in the iteration without advancing the iterator.
public int Peek() {

}

// Returns the next element in the iteration and advances the iterator.
public int Next() {

}

// Returns false if the iterator is refering to the end of the array of true
otherwise.
public bool HasNext() {

}
}
```

C Solution:

```
/*
* Problem: Peeking Iterator
* Difficulty: Medium
* Tags: array
*
* 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 Iterator {
* // Returns true if the iteration has more elements.
* bool (*hasNext)();
*
* // Returns the next element in the iteration.
* int (*next)();
* };
*/

struct PeekingIterator {

};

struct PeekingIterator* Constructor(struct Iterator* iter) {
struct PeekingIterator* piter = malloc(sizeof(struct PeekingIterator));
piter->iterator = iter;
piter->hasPeeked = false;
return piter;
}

int peek(struct PeekingIterator* obj) {

}

int next(struct PeekingIterator* obj) {

}

bool hasNext(struct PeekingIterator* obj) {

}

/**
* Your PeekingIterator struct will be instantiated and called as such:
```

```
* PeekingIterator* obj = peekingIteratorCreate(arr, arrSize);
* int param_1 = peek(obj);
* int param_2 = next(obj);
* bool param_3 = hasNext(obj);
* peekingIteratorFree(obj);
*/
```

Go Solution:

```
// Problem: Peeking Iterator
// Difficulty: Medium
// Tags: array
//
// 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

/* Below is the interface for Iterator, which is already defined for you.
*
* type Iterator struct {
*
* }
*
* func (this *Iterator) hasNext() bool {
* // Returns true if the iteration has more elements.
* }
*
* func (this *Iterator) next() int {
* // Returns the next element in the iteration.
* }
*/

type PeekingIterator struct {

}

func Constructor(iter *Iterator) *PeekingIterator {

}

func (this *PeekingIterator) hasNext() bool {
}

func (this *PeekingIterator) next() int {

}

func (this *PeekingIterator) peek() int {

}
```

Kotlin Solution:

```
// Kotlin Iterator reference:
// https://kotlinlang.org/api/latest/jvm/stdlib/kotlin.collections/-iterator/

class PeekingIterator(iterator:Iterator<Int>):Iterator<Int> {
fun peek(): Int {

}

override fun next(): Int {

}

override fun hasNext(): Boolean {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = PeekingIterator(arr)
* var param_1 = obj.next()
* var param_2 = obj.peek()
* var param_3 = obj.hasNext()
*/
```

Swift Solution:

```
// Swift IndexingIterator refernence:
// https://developer.apple.com/documentation/swift/indexingiterator

class PeekingIterator {
init(_ arr: IndexingIterator<Array<Int>>) {

}

func next() -> Int {

}

func peek() -> Int {

}

func hasNext() -> Bool {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* let obj = PeekingIterator(arr)
* let ret_1: Int = obj.next()
* let ret_2: Int = obj.peek()
* let ret_3: Bool = obj.hasNext()
*/
```

Ruby Solution:

```
# Below is the interface for Iterator, which is already defined for you.
#
# class Iterator
# def initialize(v)
#
# end
#
# def hasNext()
# Returns true if the iteration has more elements.
# end
#
```

```
# def next()
# Returns the next element in the iteration.
# end
# end

class PeekingIterator
# @param {Iterator} iter
def initialize(iter)

end

# Returns true if the iteration has more elements.
# @return {boolean}
def hasNext()

end

# Returns the next element in the iteration.
# @return {integer}
def next()

end

# Returns the next element in the iteration without advancing the iterator.
# @return {integer}
def peek()

end
end
```

PHP Solution:

```
// PHP ArrayIterator reference:
// https://www.php.net/arrayiterator

class PeekingIterator {
/**
* @param ArrayIterator $arr
*/
function __construct($arr) {
}

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

}

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

}

/**
* @return Boolean
*/
function hasNext() {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* $obj = PeekingIterator($arr);
* $ret_1 = $obj->next();
* $ret_2 = $obj->peek();
* $ret_3 = $obj->hasNext();
*/
```

Scala Solution:

```
// Scala Iterator reference:
// https://www.scala-lang.org/api/2.12.2/scala/collection/Iterator.html

class PeekingIterator(_iterator: Iterator[Int]) {
def peek(): Int = {

}
def next(): Int = {

}

def hasNext(): Boolean = {

}
}

/**
* Your PeekingIterator object will be instantiated and called as such:
* var obj = new PeekingIterator(arr)
* var param_1 = obj.next()
* var param_2 = obj.peek()
* var param_3 = obj.hasNext()
*/
```