Problem 341: Flatten Nested List Iterator

Problem Information

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

Problem Description

You are given a nested list of integers

nestedList

. Each element is either an integer or a list whose elements may also be integers or other lists. Implement an iterator to flatten it.

Implement the

NestedIterator

class:

NestedIterator(List<NestedInteger> nestedList)

Initializes the iterator with the nested list

nestedList

.

int next()

Returns the next integer in the nested list.

boolean hasNext()

Returns

true

if there are still some integers in the nested list and

false

otherwise.

Your code will be tested with the following pseudocode:

initialize iterator with nestedList res = [] while iterator.hasNext() append iterator.next() to the end of res return res

If

res

matches the expected flattened list, then your code will be judged as correct.

Example 1:

Input:

nestedList = [[1,1],2,[1,1]]

Output:

[1,1,2,1,1]

Explanation:

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,1,2,1,1].

Example 2:

Input:

nestedList = [1,[4,[6]]]

Output:

[1,4,6]

Explanation:

By calling next repeatedly until hasNext returns false, the order of elements returned by next should be: [1,4,6].

Constraints:

1 <= nestedList.length <= 500

The values of the integers in the nested list is in the range

[-10

6

, 10

6

]

.

Code Snippets

C++:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* public:
```

```
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger() const;
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger() const;
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* const vector<NestedInteger> &getList() const;
* };
*/

class NestedIterator {
public:
NestedIterator(vector<NestedInteger> &nestedList) {

}

int next() {

}

bool hasNext() {

}
};

/**
* Your NestedIterator object will be instantiated and called as such:
* NestedIterator i(nestedList);
* while (i.hasNext()) cout << i.next();
*/
```

Java:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* public interface NestedInteger {
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* public boolean isInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* public Integer getInteger();
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return empty list if this NestedInteger holds a single integer
* public List<NestedInteger> getList();
* }
*/
public class NestedIterator implements Iterator<Integer> {

public NestedIterator(List<NestedInteger> nestedList) {

}

@Override
public Integer next() {

}

@Override
public boolean hasNext() {

}
}

/**
* Your NestedIterator object will be instantiated and called as such:
* NestedIterator i = new NestedIterator(nestedList);
* while (i.hasNext()) v[f()] = i.next();
*/
```

Python3:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger:
# def isInteger(self) -> bool:
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# """
#
# def getInteger(self) -> int:
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# Return None if this NestedInteger holds a nested list
# """
#
# def getList(self) -> [NestedInteger]:
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# Return None if this NestedInteger holds a single integer
# """

class NestedIterator:
def __init__(self, nestedList: [NestedInteger]):


def next(self) -> int:


def hasNext(self) -> bool:


# Your NestedIterator object will be instantiated and called as such:
# i, v = NestedIterator(nestedList), []
# while i.hasNext(): v.append(i.next())
```

Python:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger(object):
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# Return None if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# Return None if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class NestedIterator(object):

def __init__(self, nestedList):
"""
Initialize your data structure here.
:type nestedList: List[NestedInteger]
"""



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



def hasNext(self):
```

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



# Your NestedIterator object will be instantiated and called as such:
# i, v = NestedIterator(nestedList), []
# while i.hasNext(): v.append(i.next())
```

JavaScript:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* function NestedInteger() {
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* @return {boolean}
* this.isInteger = function() {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* Return null if this NestedInteger holds a nested list
* @return {integer}
* this.getInteger = function() {
* ...
* };
*
* Return the nested list that this NestedInteger holds, if it holds a nested
list
* Return null if this NestedInteger holds a single integer
* @return {NestedInteger[]}
* this.getList = function() {
* ...
* };
* };
*/
/**
* @constructor
* @param {NestedInteger[]} nestedList
*/
var NestedIterator = function(nestedList) {

};



/**
* @this NestedIterator
* @returns {boolean}
*/
NestedIterator.prototype.hasNext = function() {

};

/**
* @this NestedIterator
* @returns {integer}
*/
NestedIterator.prototype.next = function() {

};

/**
* Your NestedIterator will be called like this:
* var i = new NestedIterator(nestedList), a = [];
* while (i.hasNext()) a.push(i.next());
*/
```

TypeScript:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* If value is provided, then it holds a single integer
* Otherwise it holds an empty nested list
* constructor(value?: number) {
* ...
* };
*
* Return true if this NestedInteger holds a single integer, rather than a
```

```
nested list.
* isInteger(): boolean {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* Return null if this NestedInteger holds a nested list
* getInteger(): number | null {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* setInteger(value: number) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* add(elem: NestedInteger) {
* ...
* };
*
* Return the nested list that this NestedInteger holds,
* or an empty list if this NestedInteger holds a single integer
* getList(): NestedInteger[] {
* ...
* };
* };
*/

class NestedIterator {
constructor(nestedList: NestedInteger[]) {

}

hasNext(): boolean {

}

next(): number {

}
```

```
}

/**
* Your ParkingSystem object will be instantiated and called as such:
* var obj = new NestedIterator(nestedList)
* var a: number[] = []
* while (obj.hasNext()) a.push(obj.next());
*/
```

C#:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* interface NestedInteger {
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool IsInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* int GetInteger();
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return null if this NestedInteger holds a single integer
* IList<NestedInteger> GetList();
* }
*/
public class NestedIterator {

public NestedIterator(IList<NestedInteger> nestedList) {

}

public bool HasNext() {

}

public int Next() {
}
}

/**
* Your NestedIterator will be called like this:
* NestedIterator i = new NestedIterator(nestedList);
* while (i.HasNext()) v[f()] = i.Next();
*/
```

C:

```
/**
* *********************************************************************
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* *********************************************************************
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool NestedIntegerIsInteger(struct NestedInteger *);
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int NestedIntegerGetInteger(struct NestedInteger *);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* struct NestedInteger **NestedIntegerGetList(struct NestedInteger *);
*
* // Return the nested list's size that this NestedInteger holds, if it holds
a nested list
* // The result is undefined if this NestedInteger holds a single integer
* int NestedIntegerGetListSize(struct NestedInteger *);
* };
*/
struct NestedIterator {

};
```

```
struct NestedIterator *nestedIterCreate(struct NestedInteger** nestedList,
int nestedListSize) {

}

bool nestedIterHasNext(struct NestedIterator *iter) {

}

int nestedIterNext(struct NestedIterator *iter) {

}

/** Deallocates memory previously allocated for the iterator */
void nestedIterFree(struct NestedIterator *iter) {

}

/**
* Your NestedIterator will be called like this:
* struct NestedIterator *i = nestedIterCreate(nestedList, nestedListSize);
* while (nestedIterHasNext(i)) printf("%d\n", nestedIterNext(i));
* nestedIterFree(i);
*/
```

Go:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* type NestedInteger struct {
* }
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* func (this NestedInteger) IsInteger() bool {}
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* // So before calling this method, you should have a check
* func (this NestedInteger) GetInteger() int {}
```

```
*
* // Set this NestedInteger to hold a single integer.
* func (n *NestedInteger) SetInteger(value int) {}
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* func (this *NestedInteger) Add(elem NestedInteger) {}
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The list length is zero if this NestedInteger holds a single integer
* // You can access NestedInteger's List element directly if you want to
modify it
* func (this NestedInteger) GetList() []*NestedInteger {}
*/

type NestedIterator struct {

}

func Constructor(nestedList []*NestedInteger) *NestedIterator {

}

func (this *NestedIterator) Next() int {

}

func (this *NestedIterator) HasNext() bool {

}
```

Kotlin:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Constructor initializes an empty nested list.
* constructor()
*
* // Constructor initializes a single integer.
* constructor(value: Int)
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* fun isInteger(): Boolean
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* fun getInteger(): Int?
*
* // Set this NestedInteger to hold a single integer.
* fun setInteger(value: Int): Unit
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* fun add(ni: NestedInteger): Unit
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return null if this NestedInteger holds a single integer
* fun getList(): List<NestedInteger>?
* }
*/

class NestedIterator(nestedList: List<NestedInteger>) {
fun next(): Int {

}

fun hasNext(): Boolean {

}
}

/**
* Your NestedIterator object will be instantiated and called as such:
* var obj = NestedIterator(nestedList)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* public func isInteger() -> Bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public func getInteger() -> Int
*
* // Set this NestedInteger to hold a single integer.
* public func setInteger(value: Int)
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public func add(elem: NestedInteger)
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public func getList() -> [NestedInteger]
* }
*/

class NestedIterator {

init(_ nestedList: [NestedInteger]) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

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

Rust:

```
// #[derive(Debug, PartialEq, Eq)]
// pub enum NestedInteger {
// Int(i32),
// List(Vec<NestedInteger>)
// }
struct NestedIterator {

}



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

fn new(nestedList: Vec<NestedInteger>) -> Self {

}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

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

```
* let obj = NestedIterator::new(nestedList);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby:

```
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
#
#class NestedInteger
# def is_integer()
# """
# Return true if this NestedInteger holds a single integer, rather than a
nested list.
# @return {Boolean}
# """
#
# def get_integer()
# """
# Return the single integer that this NestedInteger holds, if it holds a
single integer
# Return nil if this NestedInteger holds a nested list
# @return {Integer}
# """
#
# def get_list()
# """
# Return the nested list that this NestedInteger holds, if it holds a nested
list
# Return nil if this NestedInteger holds a single integer
# @return {NestedInteger[]}
# """

class NestedIterator
# @param {NestedInteger[]} nested_list
def initialize(nested_list)

end

# @return {Boolean}
def has_next
```

```
end

# @return {Integer}
def next

end
end

# Your NestedIterator will be called like this:
# i, v = NestedIterator.new(nested_list), []
# while i.has_next()
# v << i.next
# end
```

PHP:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
*
* // if value is not specified, initializes an empty list.
* // Otherwise initializes a single integer equal to value.
* function __construct($value = null)
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* function isInteger() : bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* function getInteger()
*
* // Set this NestedInteger to hold a single integer.
* function setInteger($i) : void
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* function add($ni) : void
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* function getList() : array
* }
*/

class NestedIterator {
/**
* @param NestedInteger[] $nestedList
*/
function __construct($nestedList) {

}

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

}

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

}
}

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

Scala:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* trait NestedInteger {
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* def isInteger: Boolean
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer.
* def getInteger: Int
*
* // Set this NestedInteger to hold a single integer.
* def setInteger(i: Int): Unit
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list.
* def getList: Array[NestedInteger]
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* def add(ni: NestedInteger): Unit
* }
*/

class NestedIterator(_nestedList: List[NestedInteger]) {
def next(): Int = {

}

def hasNext(): Boolean = {

}
}

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

Solutions

C++ Solution:

```
/*
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* public:
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool isInteger() const;
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int getInteger() const;
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* const vector<NestedInteger> &getList() const;
* };
*/

class NestedIterator {
public:
NestedIterator(vector<NestedInteger> &nestedList) {

}

int next() {
}

bool hasNext() {

}
};

/**
* Your NestedIterator object will be instantiated and called as such:
* NestedIterator i(nestedList);
* while (i.hasNext()) cout << i.next();
*/
```

Java Solution:

```
/**
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* public interface NestedInteger {
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* public boolean isInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* public Integer getInteger();
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return empty list if this NestedInteger holds a single integer
* public List<NestedInteger> getList();
* }
*/
public class NestedIterator implements Iterator<Integer> {

public NestedIterator(List<NestedInteger> nestedList) {

}

@Override
public Integer next() {

}

@Override
public boolean hasNext() {

}
}

/**
* Your NestedIterator object will be instantiated and called as such:
* NestedIterator i = new NestedIterator(nestedList);
* while (i.hasNext()) v[f()] = i.next();
*/
```

Python3 Solution:

```
"""
Problem: Flatten Nested List Iterator
Difficulty: Medium
Tags: tree, search, stack, queue

Approach: DFS or BFS traversal
Time Complexity: O(n) where n is number of nodes
Space Complexity: O(h) for recursion stack where h is height
"""

# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger:
# def isInteger(self) -> bool:
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# """
#
# def getInteger(self) -> int:
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# Return None if this NestedInteger holds a nested list
# """
#
# def getList(self) -> [NestedInteger]:
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# Return None if this NestedInteger holds a single integer
# """

class NestedIterator:
def __init__(self, nestedList: [NestedInteger]):


def next(self) -> int:


def hasNext(self) -> bool:


# Your NestedIterator object will be instantiated and called as such:
# i, v = NestedIterator(nestedList), []
# while i.hasNext(): v.append(i.next())
# TODO: Implement optimized solution
pass
```

Python Solution:

```
# """
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
# """
#class NestedInteger(object):
# def isInteger(self):
# """
# @return True if this NestedInteger holds a single integer, rather than a
nested list.
# :rtype bool
# """
#
# def getInteger(self):
# """
# @return the single integer that this NestedInteger holds, if it holds a
single integer
# Return None if this NestedInteger holds a nested list
# :rtype int
# """
#
# def getList(self):
# """
# @return the nested list that this NestedInteger holds, if it holds a nested
list
# Return None if this NestedInteger holds a single integer
# :rtype List[NestedInteger]
# """

class NestedIterator(object):

def __init__(self, nestedList):
"""
Initialize your data structure here.
:type nestedList: List[NestedInteger]
"""



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

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



# Your NestedIterator object will be instantiated and called as such:
# i, v = NestedIterator(nestedList), []
# while i.hasNext(): v.append(i.next())
```

JavaScript Solution:

```
/**
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* function NestedInteger() {
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* @return {boolean}
* this.isInteger = function() {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* Return null if this NestedInteger holds a nested list
* @return {integer}
* this.getInteger = function() {
* ...
* };
*
* Return the nested list that this NestedInteger holds, if it holds a nested
list
* Return null if this NestedInteger holds a single integer
* @return {NestedInteger[]}
* this.getList = function() {
* ...
* };
* };
*/
/**
* @constructor
* @param {NestedInteger[]} nestedList
*/
var NestedIterator = function(nestedList) {

};



/**
* @this NestedIterator
* @returns {boolean}
*/
NestedIterator.prototype.hasNext = function() {

};

/**
* @this NestedIterator
* @returns {integer}
*/
NestedIterator.prototype.next = function() {

};

/**
* Your NestedIterator will be called like this:
* var i = new NestedIterator(nestedList), a = [];
* while (i.hasNext()) a.push(i.next());
*/
```

TypeScript Solution:

```
/**
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* If value is provided, then it holds a single integer
* Otherwise it holds an empty nested list
* constructor(value?: number) {
* ...
* };
*
* Return true if this NestedInteger holds a single integer, rather than a
nested list.
* isInteger(): boolean {
* ...
* };
*
* Return the single integer that this NestedInteger holds, if it holds a
single integer
* Return null if this NestedInteger holds a nested list
* getInteger(): number | null {
* ...
* };
*
* Set this NestedInteger to hold a single integer equal to value.
* setInteger(value: number) {
* ...
* };
*
* Set this NestedInteger to hold a nested list and adds a nested integer elem
to it.
* add(elem: NestedInteger) {
```

```
* ...
* };
*
* Return the nested list that this NestedInteger holds,
* or an empty list if this NestedInteger holds a single integer
* getList(): NestedInteger[] {
* ...
* };
* };
*/

class NestedIterator {
constructor(nestedList: NestedInteger[]) {

}

hasNext(): boolean {

}

next(): number {

}
}

/**
* Your ParkingSystem object will be instantiated and called as such:
* var obj = new NestedIterator(nestedList)
* var a: number[] = []
* while (obj.hasNext()) a.push(obj.next());
*/
```

C# Solution:

```
/*
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* interface NestedInteger {
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool IsInteger();
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* int GetInteger();
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return null if this NestedInteger holds a single integer
* IList<NestedInteger> GetList();
* }
*/
public class NestedIterator {

public NestedIterator(IList<NestedInteger> nestedList) {

}

public bool HasNext() {

}

public int Next() {

}
}

/**
* Your NestedIterator will be called like this:
* NestedIterator i = new NestedIterator(nestedList);
* while (i.HasNext()) v[f()] = i.Next();
```

```
*/
```

C Solution:

```
/*
* Problem: Flatten Nested List Iterator
* Difficulty: Medium
* Tags: tree, search, stack, queue
*
* Approach: DFS or BFS traversal
* Time Complexity: O(n) where n is number of nodes
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* *********************************************************************
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* *********************************************************************
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* bool NestedIntegerIsInteger(struct NestedInteger *);
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* int NestedIntegerGetInteger(struct NestedInteger *);
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* struct NestedInteger **NestedIntegerGetList(struct NestedInteger *);
*
* // Return the nested list's size that this NestedInteger holds, if it holds
a nested list
* // The result is undefined if this NestedInteger holds a single integer
* int NestedIntegerGetListSize(struct NestedInteger *);
* };
*/
struct NestedIterator {
};

struct NestedIterator *nestedIterCreate(struct NestedInteger** nestedList,
int nestedListSize) {

}

bool nestedIterHasNext(struct NestedIterator *iter) {

}

int nestedIterNext(struct NestedIterator *iter) {

}

/** Deallocates memory previously allocated for the iterator */
void nestedIterFree(struct NestedIterator *iter) {

}

/**
* Your NestedIterator will be called like this:
* struct NestedIterator *i = nestedIterCreate(nestedList, nestedListSize);
* while (nestedIterHasNext(i)) printf("%d\n", nestedIterNext(i));
* nestedIterFree(i);
*/
```

Go Solution:

```
// Problem: Flatten Nested List Iterator
// Difficulty: Medium
// Tags: tree, search, stack, queue
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* type NestedInteger struct {
* }
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* func (this NestedInteger) IsInteger() bool {}
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* // So before calling this method, you should have a check
* func (this NestedInteger) GetInteger() int {}
*
* // Set this NestedInteger to hold a single integer.
* func (n *NestedInteger) SetInteger(value int) {}
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* func (this *NestedInteger) Add(elem NestedInteger) {}
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The list length is zero if this NestedInteger holds a single integer
* // You can access NestedInteger's List element directly if you want to
modify it
* func (this NestedInteger) GetList() []*NestedInteger {}
*/

type NestedIterator struct {

}

func Constructor(nestedList []*NestedInteger) *NestedIterator {

}

func (this *NestedIterator) Next() int {

}

func (this *NestedIterator) HasNext() bool {
}
```

Kotlin Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Constructor initializes an empty nested list.
* constructor()
*
* // Constructor initializes a single integer.
* constructor(value: Int)
*
* // @return true if this NestedInteger holds a single integer, rather than a
nested list.
* fun isInteger(): Boolean
*
* // @return the single integer that this NestedInteger holds, if it holds a
single integer
* // Return null if this NestedInteger holds a nested list
* fun getInteger(): Int?
*
* // Set this NestedInteger to hold a single integer.
* fun setInteger(value: Int): Unit
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* fun add(ni: NestedInteger): Unit
*
* // @return the nested list that this NestedInteger holds, if it holds a
nested list
* // Return null if this NestedInteger holds a single integer
* fun getList(): List<NestedInteger>?
* }
*/

class NestedIterator(nestedList: List<NestedInteger>) {
fun next(): Int {

}
fun hasNext(): Boolean {

}
}

/**
* Your NestedIterator object will be instantiated and called as such:
* var obj = NestedIterator(nestedList)
* var param_1 = obj.next()
* var param_2 = obj.hasNext()
*/
```

Swift Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* public func isInteger() -> Bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* public func getInteger() -> Int
*
* // Set this NestedInteger to hold a single integer.
* public func setInteger(value: Int)
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* public func add(elem: NestedInteger)
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* public func getList() -> [NestedInteger]
* }
*/
```

```
class NestedIterator {

init(_ nestedList: [NestedInteger]) {

}

func next() -> Int {

}

func hasNext() -> Bool {

}
}

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

Rust Solution:

```
// Problem: Flatten Nested List Iterator
// Difficulty: Medium
// Tags: tree, search, stack, queue
//
// Approach: DFS or BFS traversal
// Time Complexity: O(n) where n is number of nodes
// Space Complexity: O(h) for recursion stack where h is height

// #[derive(Debug, PartialEq, Eq)]
// pub enum NestedInteger {
// Int(i32),
// List(Vec<NestedInteger>)
// }
struct NestedIterator {

}
```

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

fn new(nestedList: Vec<NestedInteger>) -> Self {

}

fn next(&self) -> i32 {

}

fn has_next(&self) -> bool {

}
}

/**
* Your NestedIterator object will be instantiated and called as such:
* let obj = NestedIterator::new(nestedList);
* let ret_1: i32 = obj.next();
* let ret_2: bool = obj.has_next();
*/
```

Ruby Solution:

```
# This is the interface that allows for creating nested lists.
# You should not implement it, or speculate about its implementation
#
#class NestedInteger
# def is_integer()
# """
# Return true if this NestedInteger holds a single integer, rather than a
nested list.
# @return {Boolean}
# """
#
```

```
# def get_integer()
# """
# Return the single integer that this NestedInteger holds, if it holds a
single integer
# Return nil if this NestedInteger holds a nested list
# @return {Integer}
# """
#
# def get_list()
# """
# Return the nested list that this NestedInteger holds, if it holds a nested
list
# Return nil if this NestedInteger holds a single integer
# @return {NestedInteger[]}
# """

class NestedIterator
# @param {NestedInteger[]} nested_list
def initialize(nested_list)

end

# @return {Boolean}
def has_next

end

# @return {Integer}
def next

end
end

# Your NestedIterator will be called like this:
# i, v = NestedIterator.new(nested_list), []
# while i.has_next()
# v << i.next
# end
```

PHP Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* class NestedInteger {
*
* // if value is not specified, initializes an empty list.
* // Otherwise initializes a single integer equal to value.
* function __construct($value = null)
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* function isInteger() : bool
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer
* // The result is undefined if this NestedInteger holds a nested list
* function getInteger()
*
* // Set this NestedInteger to hold a single integer.
* function setInteger($i) : void
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* function add($ni) : void
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list
* // The result is undefined if this NestedInteger holds a single integer
* function getList() : array
* }
*/

class NestedIterator {
/**
* @param NestedInteger[] $nestedList
*/
function __construct($nestedList) {

}

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

}

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

}
}

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

Scala Solution:

```
/**
* // This is the interface that allows for creating nested lists.
* // You should not implement it, or speculate about its implementation
* trait NestedInteger {
*
* // Return true if this NestedInteger holds a single integer, rather than a
nested list.
* def isInteger: Boolean
*
* // Return the single integer that this NestedInteger holds, if it holds a
single integer.
* def getInteger: Int
*
* // Set this NestedInteger to hold a single integer.
* def setInteger(i: Int): Unit
*
* // Return the nested list that this NestedInteger holds, if it holds a
nested list.
* def getList: Array[NestedInteger]
*
* // Set this NestedInteger to hold a nested list and adds a nested integer
to it.
* def add(ni: NestedInteger): Unit
* }
*/

class NestedIterator(_nestedList: List[NestedInteger]) {
def next(): Int = {

}

def hasNext(): Boolean = {

}
}

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