

Problem 707: Design Linked List

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Design your implementation of the linked list. You can choose to use a singly or doubly linked list.

A node in a singly linked list should have two attributes:

val

and

next

.

val

is the value of the current node, and

next

is a pointer/reference to the next node.

If you want to use the doubly linked list, you will need one more attribute

prev

to indicate the previous node in the linked list. Assume all nodes in the linked list are

0-indexed

.

Implement the

MyLinkedList

class:

MyLinkedList()

Initializes the

MyLinkedList

object.

int get(int index)

Get the value of the

index

th

node in the linked list. If the index is invalid, return

-1

.

void addAtHead(int val)

Add a node of value

val

before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.

```
void addAtTail(int val)
```

Append a node of value

val

as the last element of the linked list.

```
void addAtIndex(int index, int val)
```

Add a node of value

val

before the

index

th

node in the linked list. If

index

equals the length of the linked list, the node will be appended to the end of the linked list. If

index

is greater than the length, the node

will not be inserted

.

```
void deleteAtIndex(int index)
```

Delete the

index

th

node in the linked list, if the index is valid.

Example 1:

Input

```
["MyLinkedList", "addAtHead", "addAtTail", "addAtIndex", "get", "deleteAtIndex", "get"] [[], [1], [3], [1, 2], [1], [1], [1]]
```

Output

```
[null, null, null, null, 2, null, 3]
```

Explanation

```
MyLinkedList myLinkedList = new MyLinkedList(); myLinkedList.addAtHead(1);  
myLinkedList.addAtTail(3); myLinkedList.addAtIndex(1, 2); // linked list becomes 1->2->3  
myLinkedList.get(1); // return 2 myLinkedList.deleteAtIndex(1); // now the linked list is 1->3  
myLinkedList.get(1); // return 3
```

Constraints:

$0 \leq \text{index}, \text{val} \leq 1000$

Please do not use the built-in LinkedList library.

At most

2000

calls will be made to

get

,

addAtHead

,

addAtTail

,

addAtIndex

and

deleteAtIndex

.

Code Snippets

C++:

```
class MyLinkedList {
public:
    MyLinkedList() {

    }

    int get(int index) {

    }

    void addAtHead(int val) {

    }

    void addAtTail(int val) {

    }
```

```

void addAtIndex(int index, int val) {

}

void deleteAtIndex(int index) {

}

};

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList* obj = new MyLinkedList();
 * int param_1 = obj->get(index);
 * obj->addAtHead(val);
 * obj->addAtTail(val);
 * obj->addAtIndex(index,val);
 * obj->deleteAtIndex(index);
 */

```

Java:

```

class MyLinkedList {

public MyLinkedList() {

}

public int get(int index) {

}

public void addAtHead(int val) {

}

public void addAtTail(int val) {

}

public void addAtIndex(int index, int val) {

```

```

}

public void deleteAtIndex(int index) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = new MyLinkedList();
 * int param_1 = obj.get(index);
 * obj.addAtHead(val);
 * obj.addAtTail(val);
 * obj.addAtIndex(index, val);
 * obj.deleteAtIndex(index);
 */

```

Python3:

```

class MyLinkedList:

    def __init__(self):

    def get(self, index: int) -> int:

    def addAtHead(self, val: int) -> None:

    def addAtTail(self, val: int) -> None:

    def addAtIndex(self, index: int, val: int) -> None:

    def deleteAtIndex(self, index: int) -> None:

    # Your MyLinkedList object will be instantiated and called as such:
    # obj = MyLinkedList()

```

```
# param_1 = obj.get(index)
# obj.addAtHead(val)
# obj.addAtTail(val)
# obj.addAtIndex(index, val)
# obj.deleteAtIndex(index)
```

Python:

```
class MyLinkedList(object):

    def __init__(self):

    def get(self, index):
        """
        :type index: int
        :rtype: int
        """

    def addAtHead(self, val):
        """
        :type val: int
        :rtype: None
        """

    def addAtTail(self, val):
        """
        :type val: int
        :rtype: None
        """

    def addAtIndex(self, index, val):
        """
        :type index: int
        :type val: int
        :rtype: None
        """
```

```

def deleteAtIndex(self, index):
    """
    :type index: int
    :rtype: None
    """

# Your MyLinkedList object will be instantiated and called as such:
# obj = MyLinkedList()
# param_1 = obj.get(index)
# obj.addAtHead(val)
# obj.addAtTail(val)
# obj.addAtIndex(index, val)
# obj.deleteAtIndex(index)

```

JavaScript:

```

var MyLinkedList = function() {

};

/**
 * @param {number} index
 * @return {number}
 */
MyLinkedList.prototype.get = function(index) {

};

/**
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtHead = function(val) {

};

/**
 * @param {number} val
 * @return {void}
 */

```

```

*/
MyLinkedList.prototype.addAtTail = function(val) {

};

/**
 * @param {number} index
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtIndex = function(index, val) {

};

/**
 * @param {number} index
 * @return {void}
 */
MyLinkedList.prototype.deleteAtIndex = function(index) {

};

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = new MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index, val)
 * obj.deleteAtIndex(index)
 */

```

TypeScript:

```

class MyLinkedList {
  constructor() {

  }

  get(index: number): number {

  }
}

```

```

addAtHead(val: number): void {

}

addAtTail(val: number): void {

}

addAtIndex(index: number, val: number): void {

}

deleteAtIndex(index: number): void {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = new MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index,val)
 * obj.deleteAtIndex(index)
 */

```

C#:

```

public class MyLinkedList {

    public MyLinkedList() {

    }

    public int Get(int index) {

    }

    public void AddAtHead(int val) {

```

```

    }

    public void AddAtTail(int val) {

    }

    public void AddAtIndex(int index, int val) {

    }

    public void DeleteAtIndex(int index) {

    }
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = new MyLinkedList();
 * int param_1 = obj.Get(index);
 * obj.AddAtHead(val);
 * obj.AddAtTail(val);
 * obj.AddAtIndex(index,val);
 * obj.DeleteAtIndex(index);
 */

```

C:

```

typedef struct {

} MyLinkedList;

MyLinkedList* myLinkedListCreate() {

}

int myLinkedListGet(MyLinkedList* obj, int index) {

}

```

```

void myLinkedListAddAtHead(MyLinkedList* obj, int val) {

}

void myLinkedListAddAtTail(MyLinkedList* obj, int val) {

}

void myLinkedListAddAtIndex(MyLinkedList* obj, int index, int val) {

}

void myLinkedListDeleteAtIndex(MyLinkedList* obj, int index) {

}

void myLinkedListFree(MyLinkedList* obj) {

}

/**
 * Your MyLinkedList struct will be instantiated and called as such:
 * MyLinkedList* obj = myLinkedListCreate();
 * int param_1 = myLinkedListGet(obj, index);
 *
 * myLinkedListAddAtHead(obj, val);
 *
 * myLinkedListAddAtTail(obj, val);
 *
 * myLinkedListAddAtIndex(obj, index, val);
 *
 * myLinkedListDeleteAtIndex(obj, index);
 *
 * myLinkedListFree(obj);
 */

```

Go:

```

type MyLinkedList struct {

}

```

```

func Constructor() MyLinkedList {

}

func (this *MyLinkedList) Get(index int) int {

}

func (this *MyLinkedList) AddAtHead(val int) {

}

func (this *MyLinkedList) AddAtTail(val int) {

}

func (this *MyLinkedList) AddAtIndex(index int, val int) {

}

func (this *MyLinkedList) DeleteAtIndex(index int) {

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Get(index);
 * obj.AddAtHead(val);
 * obj.AddAtTail(val);
 * obj.AddAtIndex(index,val);
 * obj.DeleteAtIndex(index);
 */

```

Kotlin:

```
class MyLinkedList() {

    fun get(index: Int): Int {

    }

    fun addAtHead(`val`: Int) {

    }

    fun addAtTail(`val`: Int) {

    }

    fun addAtIndex(index: Int, `val`: Int) {

    }

    fun deleteAtIndex(index: Int) {

    }

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(`val`)
 * obj.addAtTail(`val`)
 * obj.addAtIndex(index, `val`)
 * obj.deleteAtIndex(index)
 */
```

Swift:

```
class MyLinkedList {

    init() {
```

```

}

func get(_ index: Int) -> Int {

}

func addAtHead(_ val: Int) {

}

func addAtTail(_ val: Int) {

}

func addAtIndex(_ index: Int, _ val: Int) {

}

func deleteAtIndex(_ index: Int) {

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * let obj = MyLinkedList()
 * let ret_1: Int = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index, val)
 * obj.deleteAtIndex(index)
 */

```

Rust:

```

struct MyLinkedList {

}

/**

```

```

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

    fn new() -> Self {

    }

    fn get(&self, index: i32) -> i32 {

    }

    fn add_at_head(&self, val: i32) {

    }

    fn add_at_tail(&self, val: i32) {

    }

    fn add_at_index(&self, index: i32, val: i32) {

    }

    fn delete_at_index(&self, index: i32) {

    }
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * let obj = MyLinkedList::new();
 * let ret_1: i32 = obj.get(index);
 * obj.add_at_head(val);
 * obj.add_at_tail(val);
 * obj.add_at_index(index, val);
 * obj.delete_at_index(index);
 */

```

Ruby:

```
class MyLinkedList
def initialize()

end

=begin
:type index: Integer
:rtype: Integer
=end
def get(index)

end

=begin
:type val: Integer
:rtype: Void
=end
def add_at_head(val)

end

=begin
:type val: Integer
:rtype: Void
=end
def add_at_tail(val)

end

=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
def add_at_index(index, val)

end
```

```

=begin
:type index: Integer
:rtype: Void
=end

def delete_at_index(index)

end

end

# Your MyLinkedList object will be instantiated and called as such:
# obj = MyLinkedList.new()
# param_1 = obj.get(index)
# obj.add_at_head(val)
# obj.add_at_tail(val)
# obj.add_at_index(index, val)
# obj.delete_at_index(index)

```

PHP:

```

class MyLinkedList {
    /**
     *
     */
    function __construct() {

    }

    /**
     * @param Integer $index
     * @return Integer
     */
    function get($index) {

    }

    /**
     * @param Integer $val
     * @return NULL
     */
    function addAtHead($val) {

```

```

}

/**
 * @param Integer $val
 * @return NULL
 */
function addAtTail($val) {

}

/**
 * @param Integer $index
 * @param Integer $val
 * @return NULL
 */
function addAtIndex($index, $val) {

}

/**
 * @param Integer $index
 * @return NULL
 */
function deleteAtIndex($index) {

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * $obj = MyLinkedList();
 * $ret_1 = $obj->get($index);
 * $obj->addAtHead($val);
 * $obj->addAtTail($val);
 * $obj->addAtIndex($index, $val);
 * $obj->deleteAtIndex($index);
 */

```

Dart:

```

class MyLinkedList {

```

```

MyLinkedList() {

}

int get(int index) {

}

void addAtHead(int val) {

}

void addAtTail(int val) {

}

void addAtIndex(int index, int val) {

}

void deleteAtIndex(int index) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = MyLinkedList();
 * int param1 = obj.get(index);
 * obj.addAtHead(val);
 * obj.addAtTail(val);
 * obj.addAtIndex(index,val);
 * obj.deleteAtIndex(index);
 */

```

Scala:

```

class MyLinkedList() {

def get(index: Int): Int = {

}

}

```

```

def addAtHead(`val`: Int): Unit = {

}

def addAtTail(`val`: Int): Unit = {

}

def addAtIndex(index: Int, `val`: Int): Unit = {

}

def deleteAtIndex(index: Int): Unit = {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * val obj = new MyLinkedList()
 * val param_1 = obj.get(index)
 * obj.addAtHead(`val`)
 * obj.addAtTail(`val`)
 * obj.addAtIndex(index, `val`)
 * obj.deleteAtIndex(index)
 */

```

Elixir:

```

defmodule MyLinkedList do
  @spec init_() :: any
  def init_() do

  end

  @spec get(index :: integer) :: integer
  def get(index) do

  end
end

```

```

@spec add_at_head(val :: integer) :: any
def add_at_head(val) do

end

@spec add_at_tail(val :: integer) :: any
def add_at_tail(val) do

end

@spec add_at_index(index :: integer, val :: integer) :: any
def add_at_index(index, val) do

end

@spec delete_at_index(index :: integer) :: any
def delete_at_index(index) do

end
end

# Your functions will be called as such:
# MyLinkedList.init_()
# param_1 = MyLinkedList.get(index)
# MyLinkedList.add_at_head(val)
# MyLinkedList.add_at_tail(val)
# MyLinkedList.add_at_index(index, val)
# MyLinkedList.delete_at_index(index)

# MyLinkedList.init_ will be called before every test case, in which you can
do some necessary initializations.

```

Erlang:

```

-spec my_linked_list_init_() -> any().
my_linked_list_init_() ->
.

-spec my_linked_list_get(Index :: integer()) -> integer().
my_linked_list_get(Index) ->
.

```

```

-spec my_linked_list_add_at_head(Val :: integer()) -> any().
my_linked_list_add_at_head(Val) ->
.

-spec my_linked_list_add_at_tail(Val :: integer()) -> any().
my_linked_list_add_at_tail(Val) ->
.

-spec my_linked_list_add_at_index(Index :: integer(), Val :: integer()) ->
any().
my_linked_list_add_at_index(Index, Val) ->
.

-spec my_linked_list_delete_at_index(Index :: integer()) -> any().
my_linked_list_delete_at_index(Index) ->
.

%% Your functions will be called as such:
%% my_linked_list_init_(),
%% Param_1 = my_linked_list_get(Index),
%% my_linked_list_add_at_head(Val),
%% my_linked_list_add_at_tail(Val),
%% my_linked_list_add_at_index(Index, Val),
%% my_linked_list_delete_at_index(Index),

%% my_linked_list_init_ will be called before every test case, in which you
can do some necessary initializations.

```

Racket:

```

(define my-linked-list%
  (class object%
    (super-new)

    (init-field)

    ; get : exact-integer? -> exact-integer?
    (define/public (get index)
      )

    ; add-at-head : exact-integer? -> void?
    (define/public (add-at-head val)

```

```

)
; add-at-tail : exact-integer? -> void?
(define/public (add-at-tail val)
)
; add-at-index : exact-integer? exact-integer? -> void?
(define/public (add-at-index index val)
)
; delete-at-index : exact-integer? -> void?
(define/public (delete-at-index index)
)))

;; Your my-linked-list% object will be instantiated and called as such:
;; (define obj (new my-linked-list%))
;; (define param_1 (send obj get index))
;; (send obj add-at-head val)
;; (send obj add-at-tail val)
;; (send obj add-at-index index val)
;; (send obj delete-at-index index)

```

Solutions

C++ Solution:

```

/*
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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 MyLinkedList {
public:
    MyLinkedList() {

    }

    int get(int index) {

```

```

}

void addAtHead(int val) {

}

void addAtTail(int val) {

}

void addAtIndex(int index, int val) {

}

void deleteAtIndex(int index) {

}
};

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList* obj = new MyLinkedList();
 * int param_1 = obj->get(index);
 * obj->addAtHead(val);
 * obj->addAtTail(val);
 * obj->addAtIndex(index,val);
 * obj->deleteAtIndex(index);
 */

```

Java Solution:

```

/**
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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 MyLinkedList {

public MyLinkedList() {

}

public int get(int index) {

}

public void addAtHead(int val) {

}

public void addAtTail(int val) {

}

public void addAtIndex(int index, int val) {

}

public void deleteAtIndex(int index) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = new MyLinkedList();
 * int param_1 = obj.get(index);
 * obj.addAtHead(val);
 * obj.addAtTail(val);
 * obj.addAtIndex(index,val);
 * obj.deleteAtIndex(index);
 */

```

Python3 Solution:

```

"""
Problem: Design Linked List
Difficulty: Medium
Tags: linked_list

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 MyLinkedList:

    def __init__(self):

    def get(self, index: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class MyLinkedList(object):

    def __init__(self):

    def get(self, index):
        """
        :type index: int
        :rtype: int
        """

    def addAtHead(self, val):
        """
        :type val: int
        :rtype: None
        """

    def addAtTail(self, val):
        """

```

```

:type val: int
:rtype: None
"""

def addAtIndex(self, index, val):
    """
    :type index: int
    :type val: int
    :rtype: None
    """

def deleteAtIndex(self, index):
    """
    :type index: int
    :rtype: None
    """

# Your MyLinkedList object will be instantiated and called as such:
# obj = MyLinkedList()
# param_1 = obj.get(index)
# obj.addAtHead(val)
# obj.addAtTail(val)
# obj.addAtIndex(index,val)
# obj.deleteAtIndex(index)

```

JavaScript Solution:

```

/**
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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
 */

```

```
var MyLinkedList = function() {

};

/**
 * @param {number} index
 * @return {number}
 */
MyLinkedList.prototype.get = function(index) {

};

/**
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtHead = function(val) {

};

/**
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtTail = function(val) {

};

/**
 * @param {number} index
 * @param {number} val
 * @return {void}
 */
MyLinkedList.prototype.addAtIndex = function(index, val) {

};

/**
 * @param {number} index
 * @return {void}
 */
*/
```

```

MyLinkedList.prototype.deleteAtIndex = function(index) {

};

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = new MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index,val)
 * obj.deleteAtIndex(index)
 */

```

TypeScript Solution:

```

/**
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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 MyLinkedList {
  constructor() {

  }

  get(index: number): number {

  }

  addAtHead(val: number): void {

  }

  addAtTail(val: number): void {

```

```

}

addAtIndex(index: number, val: number): void {

}

deleteAtIndex(index: number): void {

}
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = new MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index,val)
 * obj.deleteAtIndex(index)
 */

```

C# Solution:

```

/*
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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 MyLinkedList {

    public MyLinkedList() {

    }

    public int Get(int index) {

```

```

    }

    public void AddAtHead(int val) {

    }

    public void AddAtTail(int val) {

    }

    public void AddAtIndex(int index, int val) {

    }

    public void DeleteAtIndex(int index) {

    }
}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = new MyLinkedList();
 * int param_1 = obj.Get(index);
 * obj.AddAtHead(val);
 * obj.AddAtTail(val);
 * obj.AddAtIndex(index,val);
 * obj.DeleteAtIndex(index);
 */

```

C Solution:

```

/*
 * Problem: Design Linked List
 * Difficulty: Medium
 * Tags: linked_list
 *
 * 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
 */

```

```

typedef struct {

} MyLinkedList;

MyLinkedList* myLinkedListCreate() {

}

int myLinkedListGet(MyLinkedList* obj, int index) {

}

void myLinkedListAddAtHead(MyLinkedList* obj, int val) {

}

void myLinkedListAddAtTail(MyLinkedList* obj, int val) {

}

void myLinkedListAddAtIndex(MyLinkedList* obj, int index, int val) {

}

void myLinkedListDeleteAtIndex(MyLinkedList* obj, int index) {

}

void myLinkedListFree(MyLinkedList* obj) {

}

/**
 * Your MyLinkedList struct will be instantiated and called as such:
 * MyLinkedList* obj = myLinkedListCreate();
 * int param_1 = myLinkedListGet(obj, index);
 *
 * myLinkedListAddAtHead(obj, val);

```

```

* myLinkedListAddAtTail(obj, val);

* myLinkedListAddAtIndex(obj, index, val);

* myLinkedListDeleteAtIndex(obj, index);

* myLinkedListFree(obj);
*/

```

Go Solution:

```

// Problem: Design Linked List
// Difficulty: Medium
// Tags: linked_list
//
// 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

type MyLinkedList struct {

}

func Constructor() MyLinkedList {

}

func (this *MyLinkedList) Get(index int) int {

}

func (this *MyLinkedList) AddAtHead(val int) {

}

func (this *MyLinkedList) AddAtTail(val int) {

```

```

}

func (this *MyLinkedList) AddAtIndex(index int, val int) {

}

func (this *MyLinkedList) DeleteAtIndex(index int) {

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * obj := Constructor();
 * param_1 := obj.Get(index);
 * obj.AddAtHead(val);
 * obj.AddAtTail(val);
 * obj.AddAtIndex(index,val);
 * obj.DeleteAtIndex(index);
 */

```

Kotlin Solution:

```

class MyLinkedList() {

    fun get(index: Int): Int {

    }

    fun addAtHead(`val`: Int) {

    }

    fun addAtTail(`val`: Int) {

    }

    fun addAtIndex(index: Int, `val`: Int) {

```

```

}

fun deleteAtIndex(index: Int) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * var obj = MyLinkedList()
 * var param_1 = obj.get(index)
 * obj.addAtHead(`val`)
 * obj.addAtTail(`val`)
 * obj.addAtIndex(index,`val`)
 * obj.deleteAtIndex(index)
 */

```

Swift Solution:

```

class MyLinkedList {

    init() {

    }

    func get(_ index: Int) -> Int {

    }

    func addAtHead(_ val: Int) {

    }

    func addAtTail(_ val: Int) {

    }

    func addAtIndex(_ index: Int, _ val: Int) {

```

```

}

func deleteAtIndex(_ index: Int) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * let obj = MyLinkedList()
 * let ret_1: Int = obj.get(index)
 * obj.addAtHead(val)
 * obj.addAtTail(val)
 * obj.addAtIndex(index, val)
 * obj.deleteAtIndex(index)
 */

```

Rust Solution:

```

// Problem: Design Linked List
// Difficulty: Medium
// Tags: linked_list
//
// 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

struct MyLinkedList {

}

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

impl MyLinkedList {

fn new() -> Self {

```

```

}

fn get(&self, index: i32) -> i32 {

}

fn add_at_head(&self, val: i32) {

}

fn add_at_tail(&self, val: i32) {

}

fn add_at_index(&self, index: i32, val: i32) {

}

fn delete_at_index(&self, index: i32) {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * let obj = MyLinkedList::new();
 * let ret_1: i32 = obj.get(index);
 * obj.add_at_head(val);
 * obj.add_at_tail(val);
 * obj.add_at_index(index, val);
 * obj.delete_at_index(index);
 */

```

Ruby Solution:

```

class MyLinkedList
  def initialize()

  end

```

```
=begin
:type index: Integer
:rtype: Integer
=end
def get(index)
```

```
end
```

```
=begin
:type val: Integer
:rtype: Void
=end
def add_at_head(val)
```

```
end
```

```
=begin
:type val: Integer
:rtype: Void
=end
def add_at_tail(val)
```

```
end
```

```
=begin
:type index: Integer
:type val: Integer
:rtype: Void
=end
def add_at_index(index, val)
```

```
end
```

```
=begin
:type index: Integer
:rtype: Void
=end
def delete_at_index(index)
```

```
end
```

```
end
```

```
# Your MyLinkedList object will be instantiated and called as such:
```

```
# obj = MyLinkedList.new()
```

```
# param_1 = obj.get(index)
```

```
# obj.add_at_head(val)
```

```
# obj.add_at_tail(val)
```

```
# obj.add_at_index(index, val)
```

```
# obj.delete_at_index(index)
```

PHP Solution:

```
class MyLinkedList {  
    /**  
    */  
    function __construct() {  
  
    }  
  
    /**  
    * @param Integer $index  
    * @return Integer  
    */  
    function get($index) {  
  
    }  
  
    /**  
    * @param Integer $val  
    * @return NULL  
    */  
    function addAtHead($val) {  
  
    }  
  
    /**  
    * @param Integer $val
```

```

* @return NULL
*/
function addAtTail($val) {

}

/**
* @param Integer $index
* @param Integer $val
* @return NULL
*/
function addAtIndex($index, $val) {

}

/**
* @param Integer $index
* @return NULL
*/
function deleteAtIndex($index) {

}
}

/**
* Your MyLinkedList object will be instantiated and called as such:
* $obj = MyLinkedList();
* $ret_1 = $obj->get($index);
* $obj->addAtHead($val);
* $obj->addAtTail($val);
* $obj->addAtIndex($index, $val);
* $obj->deleteAtIndex($index);
*/

```

Dart Solution:

```

class MyLinkedList {

MyLinkedList() {

}

}

```

```

int get(int index) {

}

void addAtHead(int val) {

}

void addAtTail(int val) {

}

void addAtIndex(int index, int val) {

}

void deleteAtIndex(int index) {

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * MyLinkedList obj = MyLinkedList();
 * int param1 = obj.get(index);
 * obj.addAtHead(val);
 * obj.addAtTail(val);
 * obj.addAtIndex(index,val);
 * obj.deleteAtIndex(index);
 */

```

Scala Solution:

```

class MyLinkedList() {

def get(index: Int): Int = {

}

def addAtHead(`val`: Int): Unit = {

```

```

}

def addAtTail(`val`: Int): Unit = {

}

def addAtIndex(index: Int, `val`: Int): Unit = {

}

def deleteAtIndex(index: Int): Unit = {

}

}

/**
 * Your MyLinkedList object will be instantiated and called as such:
 * val obj = new MyLinkedList()
 * val param_1 = obj.get(index)
 * obj.addAtHead(`val`)
 * obj.addAtTail(`val`)
 * obj.addAtIndex(index,`val`)
 * obj.deleteAtIndex(index)
 */

```

Elixir Solution:

```

defmodule MyLinkedList do
  @spec init_() :: any
  def init_() do

  end

  @spec get(index :: integer) :: integer
  def get(index) do

  end

  @spec add_at_head(val :: integer) :: any

```

```

def add_at_head(val) do

end

@spec add_at_tail(val :: integer) :: any
def add_at_tail(val) do

end

@spec add_at_index(index :: integer, val :: integer) :: any
def add_at_index(index, val) do

end

@spec delete_at_index(index :: integer) :: any
def delete_at_index(index) do

end

# Your functions will be called as such:
# MyLinkedList.init_()
# param_1 = MyLinkedList.get(index)
# MyLinkedList.add_at_head(val)
# MyLinkedList.add_at_tail(val)
# MyLinkedList.add_at_index(index, val)
# MyLinkedList.delete_at_index(index)

# MyLinkedList.init_ will be called before every test case, in which you can
do some necessary initializations.

```

Erlang Solution:

```

-spec my_linked_list_init_() -> any().
my_linked_list_init_() ->
.

-spec my_linked_list_get(Index :: integer()) -> integer().
my_linked_list_get(Index) ->
.

```

```

-spec my_linked_list_add_at_head(Val :: integer()) -> any().
my_linked_list_add_at_head(Val) ->
.

-spec my_linked_list_add_at_tail(Val :: integer()) -> any().
my_linked_list_add_at_tail(Val) ->
.

-spec my_linked_list_add_at_index(Index :: integer(), Val :: integer()) ->
any().
my_linked_list_add_at_index(Index, Val) ->
.

-spec my_linked_list_delete_at_index(Index :: integer()) -> any().
my_linked_list_delete_at_index(Index) ->
.

%% Your functions will be called as such:
%% my_linked_list_init_(),
%% Param_1 = my_linked_list_get(Index),
%% my_linked_list_add_at_head(Val),
%% my_linked_list_add_at_tail(Val),
%% my_linked_list_add_at_index(Index, Val),
%% my_linked_list_delete_at_index(Index),

%% my_linked_list_init_ will be called before every test case, in which you
can do some necessary initializations.

```

Racket Solution:

```

(define my-linked-list%
  (class object%
    (super-new)

    (init-field)

    ; get : exact-integer? -> exact-integer?
    (define/public (get index)
      )

    ; add-at-head : exact-integer? -> void?

```

```
(define/public (add-at-head val)
)
; add-at-tail : exact-integer? -> void?
(define/public (add-at-tail val)
)
; add-at-index : exact-integer? exact-integer? -> void?
(define/public (add-at-index index val)
)
; delete-at-index : exact-integer? -> void?
(define/public (delete-at-index index)
)))

;; Your my-linked-list% object will be instantiated and called as such:
;; (define obj (new my-linked-list%))
;; (define param_1 (send obj get index))
;; (send obj add-at-head val)
;; (send obj add-at-tail val)
;; (send obj add-at-index index val)
;; (send obj delete-at-index index)
```