

# Problem 876: Middle of the Linked List

## Problem Information

Difficulty: **Easy**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given the

head

of a singly linked list, return

the middle node of the linked list

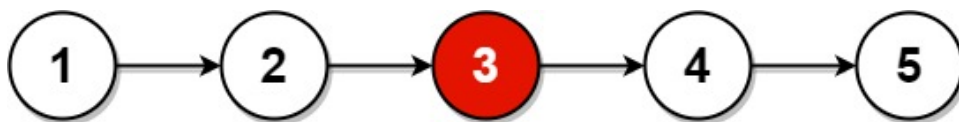
.

If there are two middle nodes, return

the second middle

node.

Example 1:



Input:

head = [1,2,3,4,5]

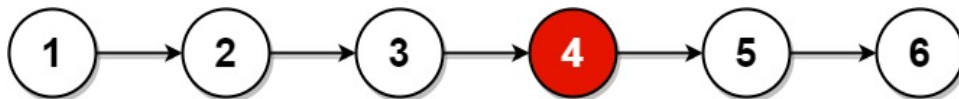
Output:

[3,4,5]

Explanation:

The middle node of the list is node 3.

Example 2:



Input:

head = [1,2,3,4,5,6]

Output:

[4,5,6]

Explanation:

Since the list has two middle nodes with values 3 and 4, we return the second one.

Constraints:

The number of nodes in the list is in the range

[1, 100]

.

$1 \leq \text{Node.val} \leq 100$

## Code Snippets

**C++:**

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
 *   ListNode *next;
 *   ListNode() : val(0), next(nullptr) {}
 *   ListNode(int x) : val(x), next(nullptr) {}
 *   ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
public:
    ListNode* middleNode(ListNode* head) {

    }
};

```

## Java:

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *   int val;
 *   ListNode next;
 *   ListNode() {}
 *   ListNode(int val) { this.val = val; }
 *   ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */
class Solution {
    public ListNode middleNode(ListNode head) {

    }
}

```

## Python3:

```

# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution:

```

```
def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:
```

## Python:

```
# Definition for singly-linked list.
# class ListNode(object):
#     def __init__(self, val=0, next=None):
#         self.val = val
#         self.next = next
class Solution(object):
    def middleNode(self, head):
        """
        :type head: Optional[ListNode]
        :rtype: Optional[ListNode]
        """
```

## JavaScript:

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *   this.val = (val===undefined ? 0 : val)
 *   this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var middleNode = function(head) {

};
```

## TypeScript:

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 *   val: number
 *   next: ListNode | null
 *   constructor(val?: number, next?: ListNode | null) {
 *     this.val = (val===undefined ? 0 : val)

```

```

* this.next = (next===undefined ? null : next)
* }
* }
*/

function middleNode(head: ListNode | null): ListNode | null {

};

```

### C#:

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public int val;
 *     public ListNode next;
 *     public ListNode(int val=0, ListNode next=null) {
 *         this.val = val;
 *         this.next = next;
 *     }
 * }
 */
public class Solution {
    public ListNode MiddleNode(ListNode head) {

    }
}

```

### C:

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* middleNode(struct ListNode* head) {

}

```

### Go:

```

/**
 * Definition for singly-linked list.
 * type ListNode struct {
 *     Val int
 *     Next *ListNode
 * }
 */
func middleNode(head *ListNode) *ListNode {

}

```

## Kotlin:

```

/**
 * Example:
 * var li = ListNode(5)
 * var v = li.`val`
 * Definition for singly-linked list.
 * class ListNode(var `val`: Int) {
 *     var next: ListNode? = null
 * }
 */
class Solution {
fun middleNode(head: ListNode?): ListNode? {

}

}

```

## Swift:

```

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     public var val: Int
 *     public var next: ListNode?
 *     public init() { self.val = 0; self.next = nil; }
 *     public init(_ val: Int) { self.val = val; self.next = nil; }
 *     public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
 * }
 */
class Solution {
fun middleNode(_ head: ListNode?) -> ListNode? {

```

```
}  
}
```

## Rust:

```
// Definition for singly-linked list.  
// #[derive(PartialEq, Eq, Clone, Debug)]  
// pub struct ListNode {  
//   pub val: i32,  
//   pub next: Option<Box<ListNode>>  
// }  
//  
// impl ListNode {  
//   #[inline]  
//   fn new(val: i32) -> Self {  
//     ListNode {  
//       next: None,  
//       val  
//     }  
//   }  
// }  
  
impl Solution {  
  pub fn middle_node(head: Option<Box<ListNode>>) -> Option<Box<ListNode>> {  
  
  }  
}
```

## Ruby:

```
# Definition for singly-linked list.  
# class ListNode  
#   attr_accessor :val, :next  
#   def initialize(val = 0, _next = nil)  
#     @val = val  
#     @next = _next  
#   end  
# end  
  
# @param {ListNode} head  
# @return {ListNode}  
def middle_node(head)
```

```
end
```

## PHP:

```
/**
 * Definition for a singly-linked list.
 * class ListNode {
 * public $val = 0;
 * public $next = null;
 * function __construct($val = 0, $next = null) {
 * $this->val = $val;
 * $this->next = $next;
 * }
 * }
 */
class Solution {

/**
 * @param ListNode $head
 * @return ListNode
 */
function middleNode($head) {

}

}
```

## Dart:

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 * int val;
 * ListNode? next;
 * ListNode([this.val = 0, this.next]);
 * }
 */
class Solution {
  ListNode? middleNode(ListNode? head) {

}

}
```



## Scala:

```
/**
 * Definition for singly-linked list.
 * class ListNode(_x: Int = 0, _next: ListNode = null) {
 *   var next: ListNode = _next
 *   var x: Int = _x
 * }
 */
object Solution {
  def middleNode(head: ListNode): ListNode = {

  }
}
```

## Elixir:

```
# Definition for singly-linked list.
#
# defmodule ListNode do
#   @type t :: %__MODULE__{
#     val: integer,
#     next: ListNode.t() | nil
#   }
#   defstruct val: 0, next: nil
# end

defmodule Solution do
  @spec middle_node(head :: ListNode.t | nil) :: ListNode.t | nil
  def middle_node(head) do

  end
end
```

## Erlang:

```
% Definition for singly-linked list.
%
% -record(list_node, {val = 0 :: integer(),
%   next = null :: 'null' | #list_node{}}).

-spec middle_node(Head :: #list_node{} | null) -> #list_node{} | null.
middle_node(Head) ->
```

.

## Racket:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
  (val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
  (list-node val #f))

|#

(define/contract (middle-node head)
  (-> (or/c list-node? #f) (or/c list-node? #f))
  )
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Middle of the Linked List
 * Difficulty: Easy
 * Tags: array, linked_list
 *
 * 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
 */

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *   int val;
```

```

* ListNode *next;
* ListNode() : val(0), next(nullptr) {}
* ListNode(int x) : val(x), next(nullptr) {}
* ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/

class Solution {
public:
    ListNode* middleNode(ListNode* head) {

    }
};

```

## Java Solution:

```

/**
 * Problem: Middle of the Linked List
 * Difficulty: Easy
 * Tags: array, linked_list
 *
 * 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
 */

/**
 * Definition for singly-linked list.
 * public class ListNode {
 *     int val;
 *     ListNode next;
 *     ListNode() {
 * // TODO: Implement optimized solution
 *     return 0;
 *     }
 *     ListNode(int val) { this.val = val; }
 *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
 * }
 */

class Solution {
    public ListNode middleNode(ListNode head) {

```

```
}  
}
```

### Python3 Solution:

```
"""  
Problem: Middle of the Linked List  
Difficulty: Easy  
Tags: array, linked_list  
  
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  
"""  
  
# Definition for singly-linked list.  
# class ListNode:  
#     def __init__(self, val=0, next=None):  
#         self.val = val  
#         self.next = next  
class Solution:  
    def middleNode(self, head: Optional[ListNode]) -> Optional[ListNode]:  
        # TODO: Implement optimized solution  
        pass
```

### Python Solution:

```
# Definition for singly-linked list.  
# class ListNode(object):  
#     def __init__(self, val=0, next=None):  
#         self.val = val  
#         self.next = next  
class Solution(object):  
    def middleNode(self, head):  
        """  
        :type head: Optional[ListNode]  
        :rtype: Optional[ListNode]  
        """
```

### JavaScript Solution:

```

/**
 * Problem: Middle of the Linked List
 * Difficulty: Easy
 * Tags: array, linked_list
 *
 * 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
 */

/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *   this.val = (val===undefined ? 0 : val)
 *   this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {ListNode}
 */
var middleNode = function(head) {

};

```

## TypeScript Solution:

```

/**
 * Problem: Middle of the Linked List
 * Difficulty: Easy
 * Tags: array, linked_list
 *
 * 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
 */

/**
 * Definition for singly-linked list.
 * class ListNode {
 *   val: number
 *   next: ListNode | null

```

```

* constructor(val?: number, next?: ListNode | null) {
* this.val = (val===undefined ? 0 : val)
* this.next = (next===undefined ? null : next)
* }
* }
*/

function middleNode(head: ListNode | null): ListNode | null {

};

```

### C# Solution:

```

/*
* Problem: Middle of the Linked List
* Difficulty: Easy
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* public class ListNode {
* public int val;
* public ListNode next;
* public ListNode(int val=0, ListNode next=null) {
* this.val = val;
* this.next = next;
* }
* }
*/

public class Solution {
public ListNode MiddleNode(ListNode head) {

}

}

```

### C Solution:

```

/*
* Problem: Middle of the Linked List
* Difficulty: Easy
* Tags: array, linked_list
*
* 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
*/

/**
* Definition for singly-linked list.
* struct ListNode {
*   int val;
*   struct ListNode *next;
* };
*/
struct ListNode* middleNode(struct ListNode* head) {

}

```

## Go Solution:

```

// Problem: Middle of the Linked List
// Difficulty: Easy
// Tags: array, linked_list
//
// 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

/**
* Definition for singly-linked list.
* type ListNode struct {
*   Val int
*   Next *ListNode
* }
*/
func middleNode(head *ListNode) *ListNode {

}

```

## Kotlin Solution:

```
/**
 * Example:
 * var li = ListNode(5)
 * var v = li.`val`
 * Definition for singly-linked list.
 * class ListNode(var `val`: Int) {
 *   var next: ListNode? = null
 * }
 */
class Solution {
    fun middleNode(head: ListNode?): ListNode? {

    }
}
```

## Swift Solution:

```
/**
 * Definition for singly-linked list.
 * public class ListNode {
 *   public var val: Int
 *   public var next: ListNode?
 *   public init() { self.val = 0; self.next = nil; }
 *   public init(_ val: Int) { self.val = val; self.next = nil; }
 *   public init(_ val: Int, _ next: ListNode?) { self.val = val; self.next =
next; }
 * }
 */
class Solution {
    func middleNode(_ head: ListNode?) -> ListNode? {

    }
}
```

## Rust Solution:

```
// Problem: Middle of the Linked List
// Difficulty: Easy
// Tags: array, linked_list
//
// 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

// Definition for singly-linked list.
// #[derive(PartialEq, Eq, Clone, Debug)]
// pub struct ListNode {
//     pub val: i32,
//     pub next: Option<Box<ListNode>>
// }
//
// impl ListNode {
//     #[inline]
//     fn new(val: i32) -> Self {
//         ListNode {
//             next: None,
//             val
//         }
//     }
// }
impl Solution {
    pub fn middle_node(head: Option<Box<ListNode>>) -> Option<Box<ListNode>> {

    }
}

```

### Ruby Solution:

```

# Definition for singly-linked list.
# class ListNode
#   attr_accessor :val, :next
#   def initialize(val = 0, _next = nil)
#     @val = val
#     @next = _next
#   end
# end
# @param {ListNode} head
# @return {ListNode}
def middle_node(head)

end

```

### PHP Solution:

```
/**
 * Definition for a singly-linked list.
 * class ListNode {
 * public $val = 0;
 * public $next = null;
 * function __construct($val = 0, $next = null) {
 * $this->val = $val;
 * $this->next = $next;
 * }
 * }
 */
class Solution {

/**
 * @param ListNode $head
 * @return ListNode
 */
function middleNode($head) {

}

}
```

### Dart Solution:

```
/**
 * Definition for singly-linked list.
 * class ListNode {
 * int val;
 * ListNode? next;
 * ListNode([this.val = 0, this.next]);
 * }
 */
class Solution {
  ListNode? middleNode(ListNode? head) {

  }

}
```

### Scala Solution:

```

/**
 * Definition for singly-linked list.
 * class ListNode(_x: Int = 0, _next: ListNode = null) {
 *   var next: ListNode = _next
 *   var x: Int = _x
 * }
 */
object Solution {
  def middleNode(head: ListNode): ListNode = {

  }
}

```

### Elixir Solution:

```

# Definition for singly-linked list.
#
# defmodule ListNode do
#   @type t :: %__MODULE__{
#     val: integer,
#     next: ListNode.t() | nil
#   }
#   defstruct val: 0, next: nil
# end

defmodule Solution do
  @spec middle_node(head :: ListNode.t | nil) :: ListNode.t | nil
  def middle_node(head) do

  end
end

```

### Erlang Solution:

```

%% Definition for singly-linked list.
%%
%% -record(list_node, {val = 0 :: integer(),
%%   next = null :: 'null' | #list_node{}}).

-spec middle_node(Head :: #list_node{} | null) -> #list_node{} | null.
middle_node(Head) ->
.

```

## Racket Solution:

```
; Definition for singly-linked list:
#|

; val : integer?
; next : (or/c list-node? #f)
(struct list-node
  (val next) #:mutable #:transparent)

; constructor
(define (make-list-node [val 0])
  (list-node val #f))

|#

(define/contract (middle-node head)
  (-> (or/c list-node? #f) (or/c list-node? #f))
  )
```