leetcode.com

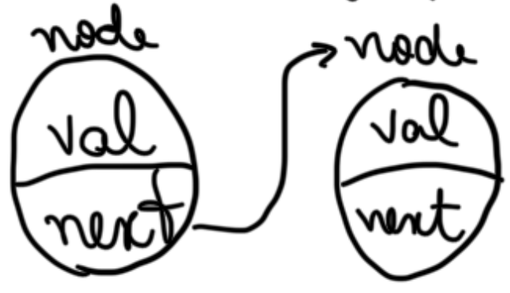
singly-linked list:

```
struct ListNode {
    int val;
    ListNode *next;
};
```

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.

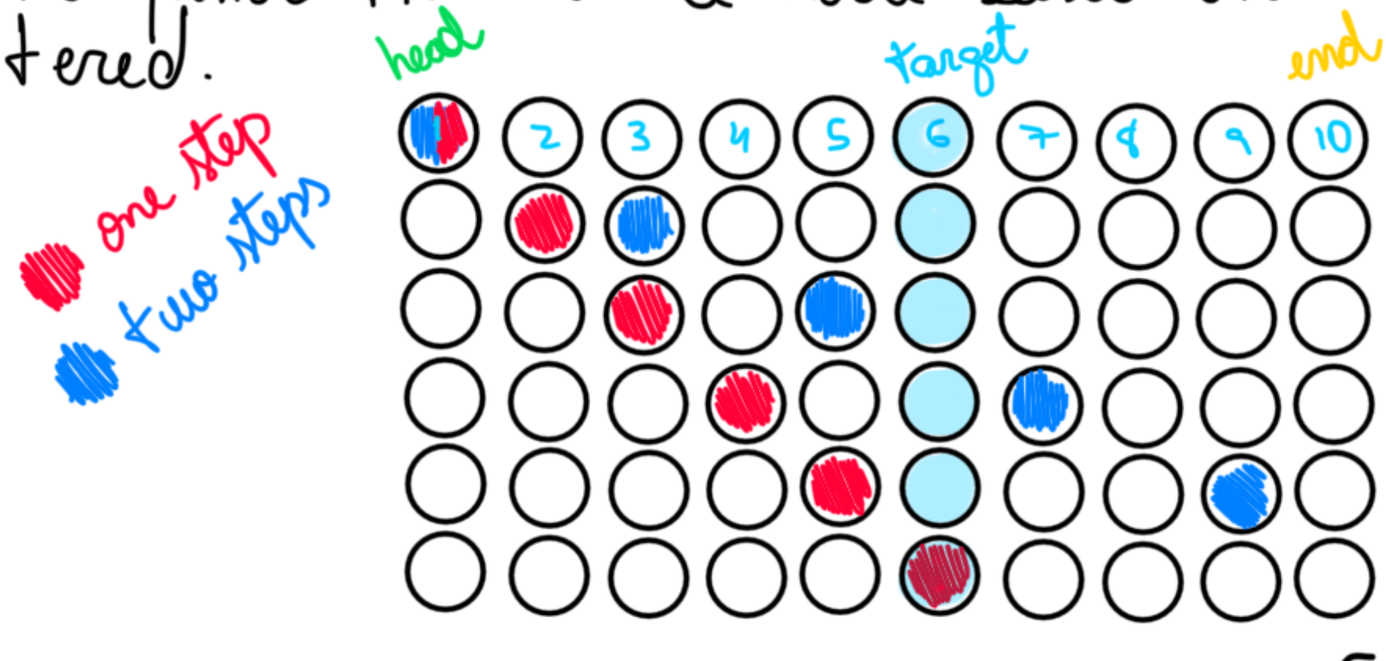
We can only know the size of a linked list by traversing it.
The "next" attribute (?) is a pointer to another node!
We can access the next node with this syntax:

(*node).next
or
node → next



To find the middle node, we can use two pointers: one to

find the last node (which will point to null) and one to find the middle node based on the last node encountered.



→ end pointer cannot take two steps, so we know the next step is the end of the linked list started at head.

Then, we can return the node → next node.

→ If blue node can't even take one step, then we must return node:

return type — name of the f(x) — type of the parameter

```
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode* endNode = head;
    struct ListNode* middleNode = head;
    while(endNode->next != null && endNode->next->next != null) {
        endNode = endNode->next->next;
        middleNode = middleNode->next;
    };
    if(endNode->next == null) {
        return middleNode;
    } else {
        return middleNode->next;
    }
}
```

&& → ~~and~~

NULL → ~~null~~

NULL → ~~null~~

↳ My strategy works, but it takes an

extra step.

The way I designed the algorithm, the end node will never be NULL, because my while condition is checking if I can move to a valid node.

But, if I don't check it and only check if the end node is either NULL or not, the middle node will always be in the middle of the linked list, regardless if it is an even or odd linked list.

So, the most graceful solution would be:

```
struct ListNode* middleNode(struct ListNode* head) {
    struct ListNode* middle = head;
    struct ListNode* end = head;
    while (end != NULL && end->next != NULL) {
        middle = middle->next;
        end = end->next->next;
```

```
        }
        return middle;
    }
```