

141. [Detect a cycle in a linked list](#)

```
class Solution {  
  
public:  
  
    bool hasCycle(ListNode *head) {  
  
        ListNode* slow = head;  
  
        ListNode* fast = head;  
  
        while (fast != nullptr && fast->next != nullptr) {  
  
            slow = slow->next;  
  
            fast = fast->next->next;  
  
            if (slow == fast) return true;  
  
        }  
  
        return false;  
  
    }  
  
};
```

The screenshot displays the LeetCode submission interface for the problem "Detect a cycle in a linked list". The submission is accepted, with a runtime of 8 ms and a memory usage of 11.74 MB. The code is written in C++ and defines a ListNode struct and a Solution class with a hasCycle method. The hasCycle method uses a slow and fast pointer approach to detect a cycle in the linked list.

Runtime Performance:

Runtime	Memory
8 ms	11.74 MB

Code:

```
/**  
 * Definition for singly-linked list.  
 * struct ListNode {  
 *     int val;  
 *     ListNode *next;  
 *     ListNode(int x) : val(x), next(NULL) {}  
 * };  
 */  
class Solution {  
public:  
    bool hasCycle(ListNode *head) {  
        ListNode* slow = head;  
        ListNode* fast = head;  
        while (fast != nullptr && fast->next != nullptr) {  
            slow = slow->next;  
            fast = fast->next->next;  
            if (slow == fast) return true;  
        }  
        return false;  
    }  
};
```

Testcase:

Case	head	pos
Case 1	[3,2,0,-4]	1