EECS 281 Lab 3 Bonus Written Problem

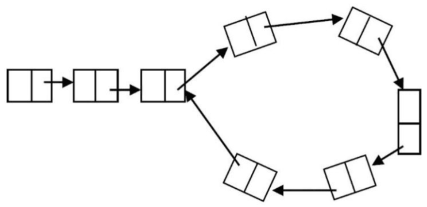
~ UNGRADED ~

BW-3. Linked List Cycle

Consider the following definition of a singly-linked list:

```
    struct ListNode {
       int val;
       ListNode *next;
       ListNode(int x) : val(x), next(nullptr) {}
    };
```

You are given the head pointer of a singly-linked list. Write a function that determines if the list contains a cycle (or loop), where a node's `next` pointer points back to a previous node in the list:



Complete this function with a runtime complexity of Θ(*n*) and an additional space complexity of Θ(1), where *n* represents the number of elements in the singly-linked list. You are limited to **15 lines of code** (not including function headers, comments, or braces on a single line).

```
// Bonus Written Problem (Lab 3): Linked List Cycle

bool hasCycle(ListNode *head) {
   // use the two pointer (runner) technique, where one moves faster than the other
   // if the fast pointer ends up catching up with the slow pointer, there must be a cycle!
   ListNode *slow = head;
   ListNode *fast = head;
   while (fast && fast->next) {
      slow = slow->next;
      fast = fast->next->next;
      if (fast == slow) {
         return true;
      }
   }
   return false;
}
```