```
int DoesLinkedListHasLoop(struct ListNode * head) {
  struct ListNode *slowPtr = head, *fastPtr = head;
  while (slowPtr && fastPtr && fastPtr→next) {
    slowPtr = slowPtr→next;
    fastPtr = fastPtr→next→next;
    if (slowPtr == fastPtr)
      return 1;
  }
  return 0;
}
```

Time Complexity: O(*n*). Space Complexity: O(1).

Problem-10 are given a pointer to the first element of a linked list *L*. There are two possibilities for *L:* it either ends (snake) or its last element points back to one of the earlier elements in the list (snail). Give an algorithm that tests whether a given list *L* is a snake or a snail.

Solution: It is the same as Problem-6.

Problem-11 Check whether the given linked list is NULL-terminated or not. If there is a cycle find the start node of the loop.

Solution: The solution is an extension to the solution in Problem-9. After finding the loop in the linked list, we initialize the *slowPtr* to the head of the linked list. From that point onwards both *slowPtr* and *fastPtr* move only one node at a time. The point at which they meet is the start of the loop. Generally we use this method for removing the loops.