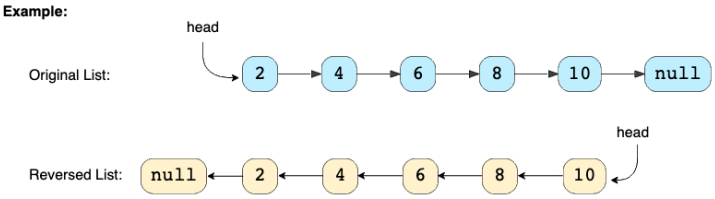
Reverse a LinkedList (easy)

We'll cover the following

Problem Statement

Given the head of a Singly LinkedList, reverse the LinkedList. Write a function to return the new head of the reversed LinkedList.



Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

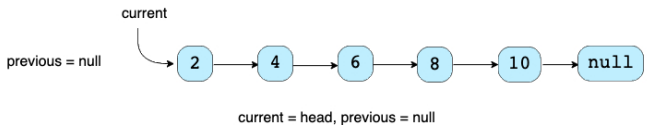
```
class Node {
  constructor(value, next=null){
    this.value = value;
    this.next = next;
  }

  get_list() {
    result = "";
    temp = this;
    while (temp !== null) {
      result += temp.value + " ";
      temp = temp.next;
    }
    return result;
  }
};


const reverse = function(head) {
  // TODO: Write your code here
  return head;
};

head = new Node(2);
head.next = new Node(4);
head.next.next = new Node(6);
head.next.next.next = new Node(8);
head.next.next.next.next = new Node(10);
```

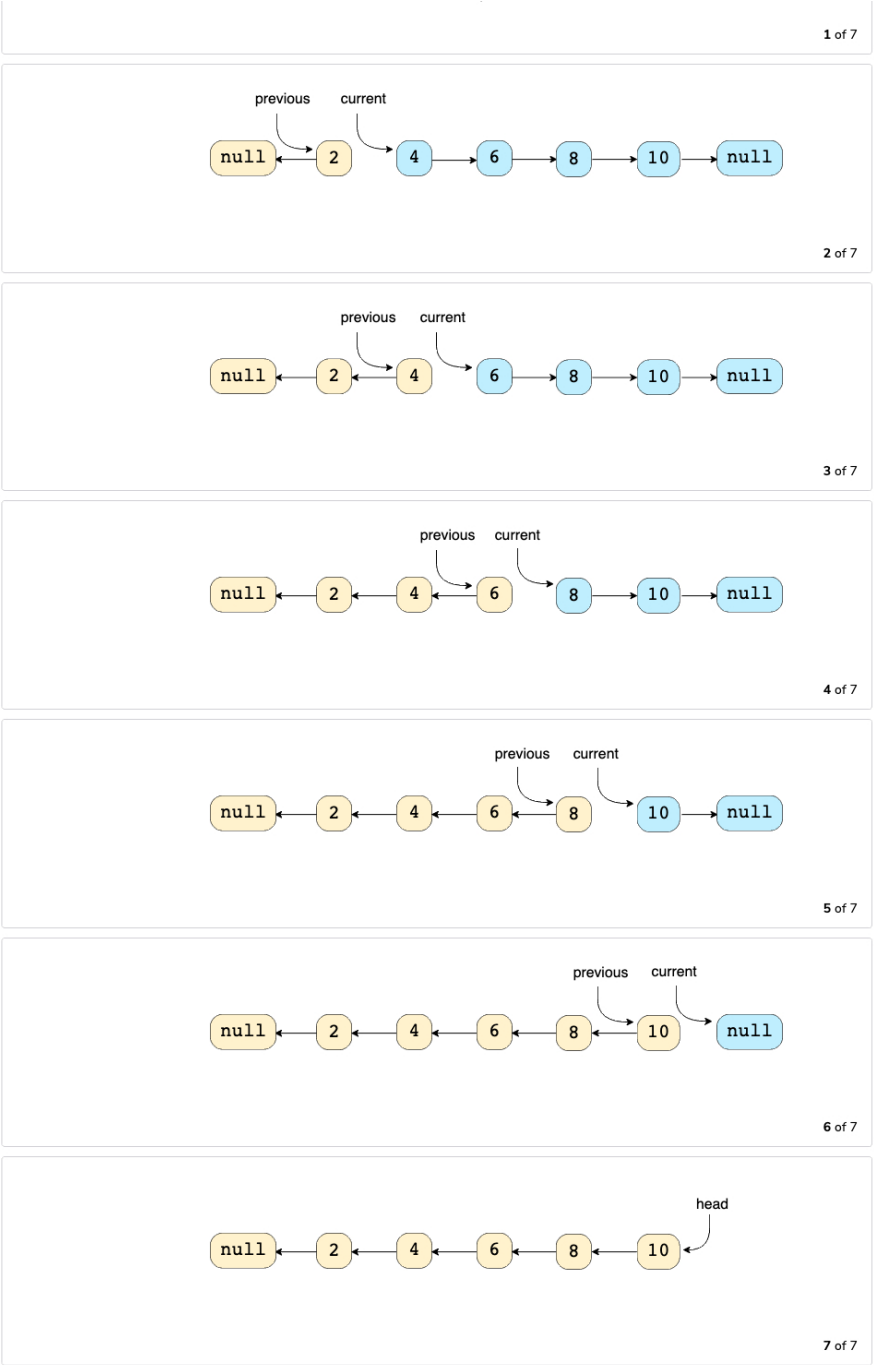
RUN SAVE RESET

Solution

To reverse a LinkedList, we need to reverse one node at a time. We will start with a variable `current` which will initially point to the head of the LinkedList and a variable `previous` which will point to the previous node that we have processed; initially `previous` will point to `null`.

In a stepwise manner, we will reverse the `current` node by pointing it to the `previous` before moving on to the next node. Also, we will update the `previous` to always point to the previous node that we have processed. Here is the visual representation of our algorithm:





Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
class Node {
  constructor(value, next = null) {
    this.value = value;
    this.next = next;
  }

  print_list() {
    let temp = this;
    while (temp !== null) {
      process.stdout.write(`${temp.value} `);
      temp = temp.next;
    }
    console.log();
  }
}


function reverse(head) {
  let current = head,
    previous = null;
  while (current !== null) {
    next = current.next; // temporarily store the next node
    current.next = previous; // reverse the current node
```

```
    previous = current; // before we move to the next node, point previous to the current node
    current = next; // move on the next node
  }
  return previous;
}
```

RUN SAVE RESET

Time complexity

The time complexity of our algorithm will be $O(N)$ where 'N' is the total number of nodes in the LinkedList.

Space complexity

We only used constant space, therefore, the space complexity of our algorithm is $O(1)$.