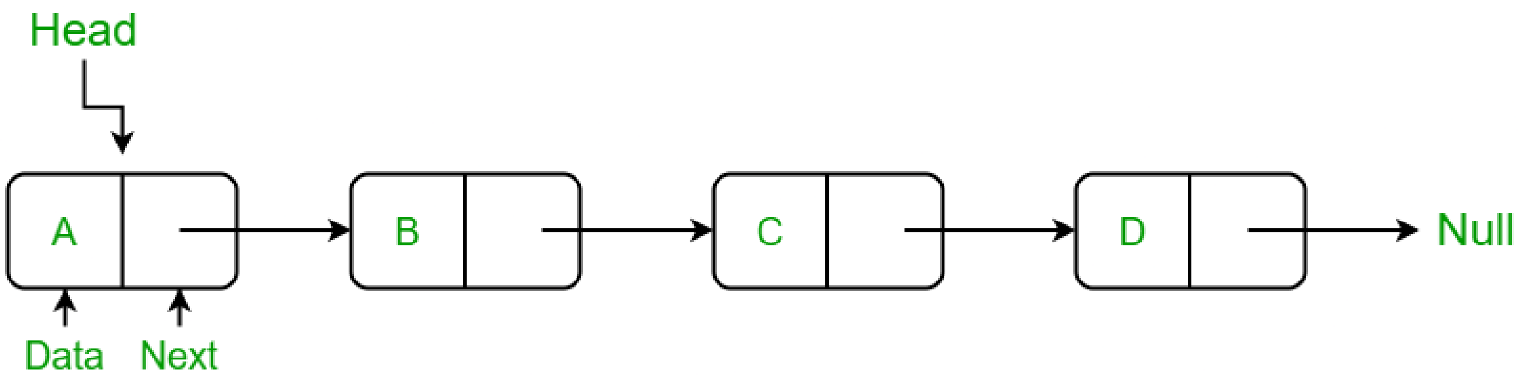
Palindrome Linked List

Given a singly linked list of characters, write a function that returns true if the given list is a palindrome, else false.



Examples:

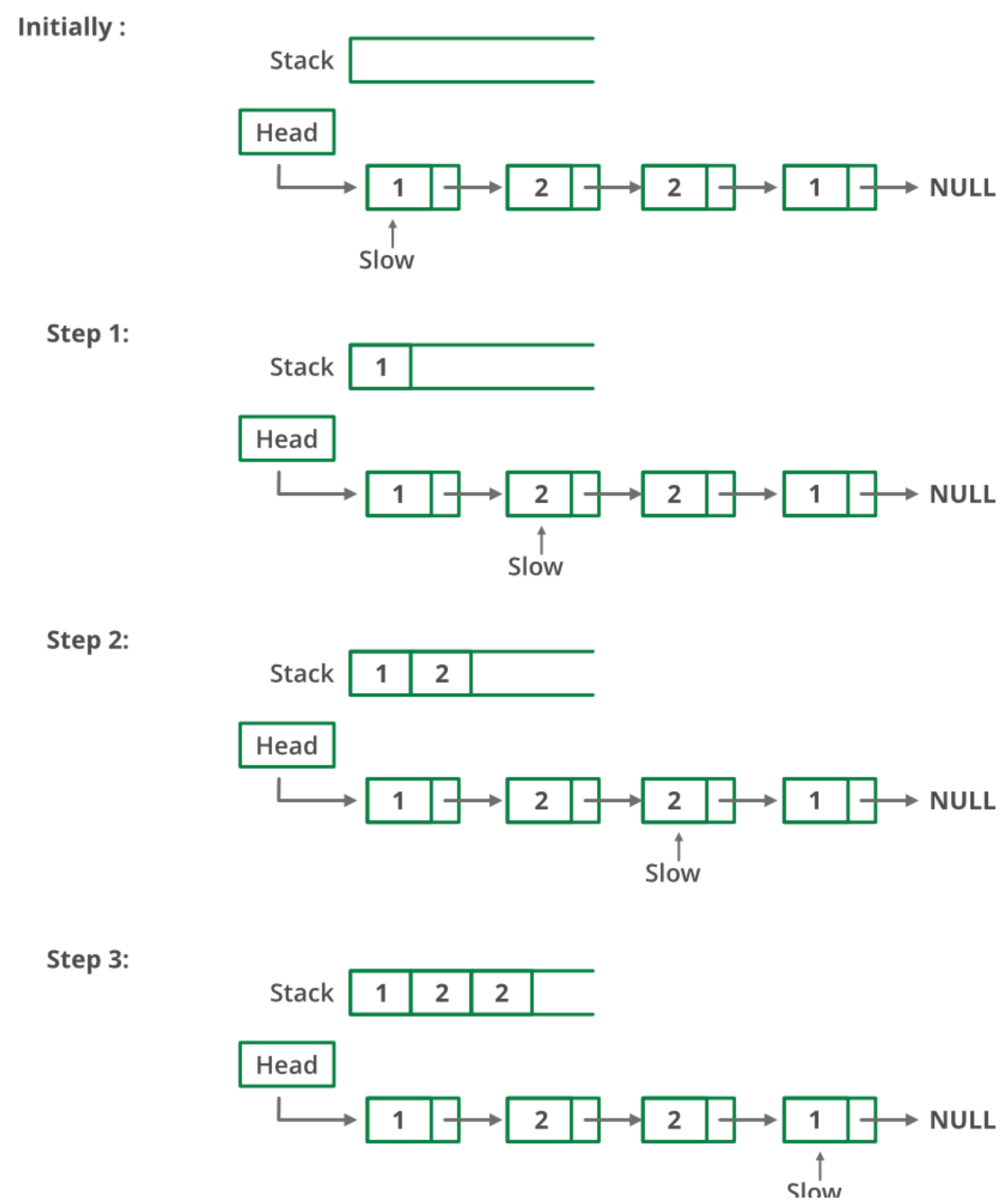
> ***Input:*** *R->A->D->A->R->NULL*
> ***Output:*** *Yes*
>
> ***Input:*** *C->O->D->E->NULL*
> ***Output:*** *No*

Check if a Singly Linked List is Palindrome using Stack:

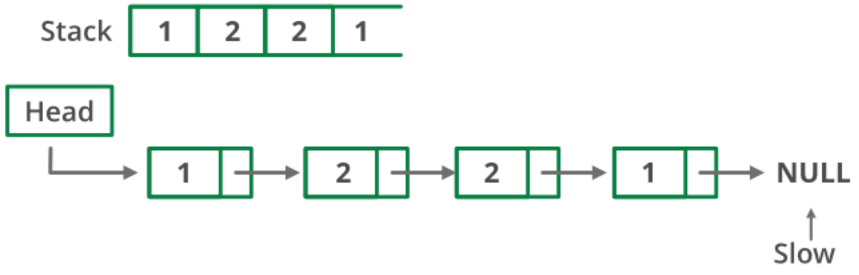
> *The idea is to use a stack and push all the nodes into the stack, then again iterate over the linked list to validate if the linked list is palindrome or not.*

Below image is a dry run of the above approach:

Dash
All
Articles
Videos
Problems
Quiz
Contest

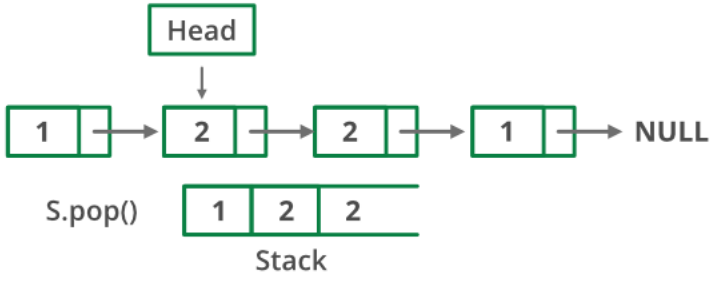

Initially :
Stack
Head
1
2
2
1
NULL
Slow
Step 1:
Stack
1
Head
1
2
2
1
NULL
Slow
Step 2:
Stack
1
2
Head
1
2
2
1
NULL
Slow
Step 3:
Stack
1
2
2
Head
1
2
2
1
NULL
Slow

Step 4:



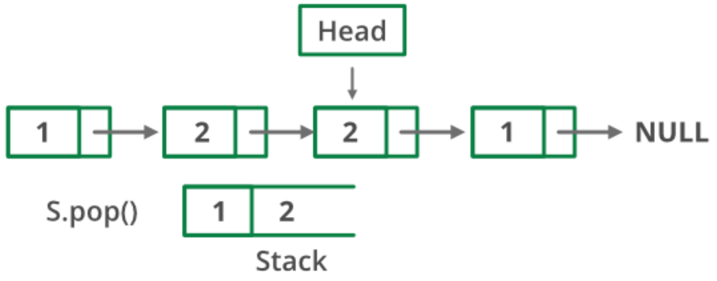
Step 5:

S.top = = head → data



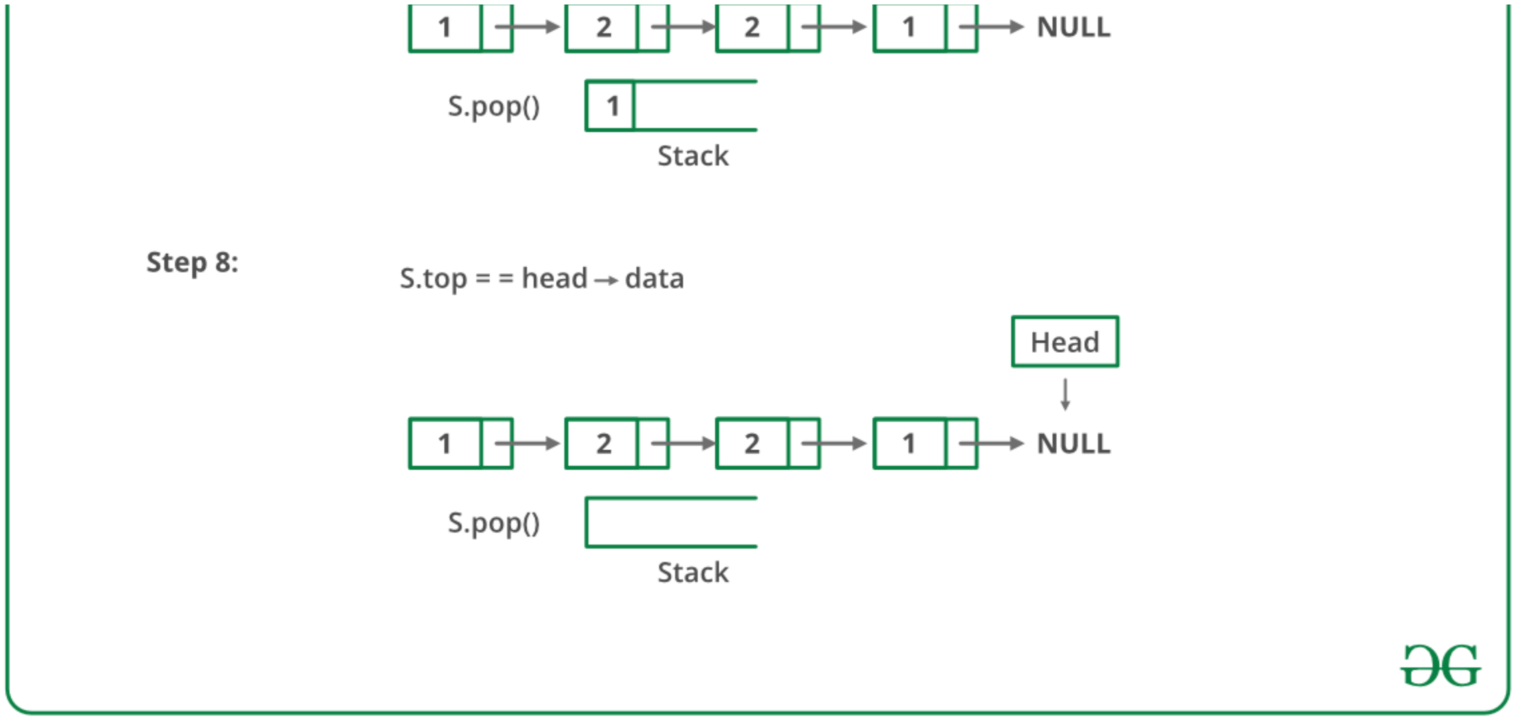
Step 6:

S.top = = head → data



Step 7:

S.top = = head → data



Follow the steps below to solve the problem:

- A simple solution is to use a stack of list nodes. This mainly involves three steps.
- Traverse the given list from head to tail and push every visited node to stack.
- Traverse the list again. For every visited node, pop a node from the stack and compare data of popped node with the currently visited node.
- If all nodes matched, then return true, else false.

Below is the implementation of the above approach :

C++ Java

```
/* Java program to check if linked list is palindrome
 * recursively */
import java.util.*;
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
class linkedList {
    public static void main(String args[])
    {
        Node one = new Node(1);
        Node two = new Node(2);
        Node three = new Node(3);
        Node four = new Node(4);
        Node five = new Node(3);
        Node six = new Node(2);
        Node seven = new Node(1);
        one.ptr = two;
        two.ptr = three;
        three.ptr = four;
        four.ptr = five;
        five.ptr = six;
        six.ptr = seven;
        boolean condition = isPalindrome(one);
        System.out.println("isPalidrome :" + condition);
    }
    static boolean isPalindrome(Node head)
    {

        Node slow = head;
        boolean ispalin = true;
        Stack<Integer> stack = new Stack<Integer>();

        while (slow != null) {
            stack.push(slow.data);
```

```
                slow = slow.ptr;
            }

            while (head != null) {

                int i = stack.pop();
                if (head.data == i) {
                    ispalin = true;
                }
                else {
                    ispalin = false;
                    break;
                }
                head = head.ptr;
            }
            return ispalin;
        }
}

class Node {
    int data;
    Node ptr;
    Node(int d)
    {
        ptr = null;
        data = d;
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

Output

```
isPalindrome is true
```

Time complexity: O(N), Iterating over the linked list of size N.
Auxiliary Space: O(N), Using an auxiliary stack

Check if a Singly Linked List is Palindrome by Reversing the Linked List:

> *The idea is to first reverse the second half part of the linked list and then check whether the list is palindrome or not.*

Follow the steps below to solve the problem:

- Get the middle of the linked list.
- Reverse the second half of the linked list.
- Check if the first half and second half are identical.
- Construct the original linked list by reversing the second half again and attaching it back to the first half

Below is the implementation of the above approach:

C++ Java

```
 /* Java program to check if linked list is palindrome */

class LinkedList {
```

```
Node head; // head of list
Node slow_ptr, fast_ptr, second_half;

/* Linked list Node*/
class Node {
    char data;
    Node next;

    Node(char d)
    {
        data = d;
        next = null;
    }
}

/* Function to check if given linked list is
palindrome or not */
boolean isPalindrome(Node head)
{
    slow_ptr = head;
    fast_ptr = head;
    Node prev_of_slow_ptr = head;
    Node midnode = null; // To handle odd size list
    boolean res = true; // initialize result

    if (head != null && head.next != null) {
        /* Get the middle of the list. Move slow_ptr by
        1 and fast_ptr by 2, slow_ptr will have the
        middle node */
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
while (fast_ptr != null
    && fast_ptr.next != null) {
    fast_ptr = fast_ptr.next.next;

    /*We need previous of the slow_ptr for
    linked lists with odd elements */
    prev_of_slow_ptr = slow_ptr;
    slow_ptr = slow_ptr.next;
}

/* fast_ptr would become NULL when there are
even elements in the list and not NULL for
odd elements. We need to skip the middle node
for odd case and store it somewhere so that
we can restore the original list */
if (fast_ptr != null) {
    midnode = slow_ptr;
    slow_ptr = slow_ptr.next;
}

// Now reverse the second half and compare it
// with first half
second_half = slow_ptr;
prev_of_slow_ptr.next
    = null; // NULL terminate first half
reverse(); // Reverse the second half
res = compareLists(head,
                   second_half); // compare
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
            /* Construct the original list back */
            reverse(); // Reverse the second half again

            if (midnode != null) {
                // If there was a mid node (odd size case)
                // which was not part of either first half
                // or second half.
                prev_of_slow_ptr.next = midnode;
                midnode.next = second_half;
            }
            else
                prev_of_slow_ptr.next = second_half;
        }
        return res;
    }

    /* Function to reverse the linked list Note that this
    function may change the head */
    void reverse()
    {
        Node prev = null;
        Node current = second_half;
        Node next;
        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
        second_half = prev;
    }

    /* Function to check if two input lists have same data*/
    boolean compareLists(Node head1, Node head2)
    {
        Node temp1 = head1;
        Node temp2 = head2;

        while (temp1 != null && temp2 != null) {
            if (temp1.data == temp2.data) {
                temp1 = temp1.next;
                temp2 = temp2.next;
            }
            else
                return false;
        }

        /* Both are empty return 1*/
        if (temp1 == null && temp2 == null)
            return true;

        /* Will reach here when one is NULL
        and other is not */
        return false;
    }

    /* Push a node to linked list. Note that this function
    changes the head */
    public void push(char new_data)
    {
        /* Allocate the Node &
        Put in the data */
        Node new_node = new Node(new_data);

        /* link the old list of the new one */

        /* Move the head to point to new Node */
        head = new_node;
    }

    // A utility function to print a given linked list
    void printList(Node ptr)
    {
        while (ptr != null) {
            System.out.print(ptr.data + "->");
            ptr = ptr.next;
        }
        System.out.println("NULL");
    }

    /* Driver program to test the above functions */
    public static void main(String[] args)
    {

        /* Start with the empty list */
        LinkedList llist = new LinkedList();
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
        char str[] = { 'a', 'b', 'a', 'c', 'a', 'b', 'a' };
        String string = new String(str);
        for (int i = 0; i < 7; i++) {
            llist.push(str[i]);
        }
        if (llist.isPalindrome(llist.head) != false) {
            System.out.println("Is Palindrome");
            System.out.println("");
        }
        else {
            System.out.println("Not Palindrome");
            System.out.println("");
        }
    }
}
```

Output

```
Is Palindrome
```

Time Complexity: O(N),
Auxiliary Space: O(1)

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.

Dash

All

Articles

Videos

Problems

Quiz

Contest