Linked List Data Structure

[**Data Structure and Algorithms Course**](https://practice.geeksforgeeks.org/courses/dsa-self-paced?utm_source=Google&utm_medium=Header+Link+Click&utm_campaign=DSA+Course+Tracker&utm_id=DSA-Course-Tracker&utm_term=DSA+Course+Promo&utm_content=Course+Page)

[**Practice Problems on Linked List**](https://practice.geeksforgeeks.org/explore/?category%5B%5D=Linked%20List&page=1&category%5B%5D=Linked%20List)

[**Recent Articles on Linked List**](https://www.geeksforgeeks.org/category/linked-list/)

**What is Linked List**

*A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers as shown in the below image:*



In simple words, a linked list consists of nodes where each node contains a data field and a reference(link) to the next node in the list.

**Topics :**

* [Introduction](https://www.geeksforgeeks.org/data-structures/linked-list/?ref=ghm#introduction)
* [Types of Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/?ref=ghm#types)
* [Basic Operations](https://www.geeksforgeeks.org/data-structures/linked-list/?ref=ghm#basicop)
* [Standard problem on Linked List](https://www.geeksforgeeks.org/data-structures/linked-list/?ref=ghm#standardp)

**Introduction:**

1. [What is Linked List](https://www.geeksforgeeks.org/what-is-linked-list/)
2. [Introduction to Linked List – Data Structure and Algorithm Tutorials](https://www.geeksforgeeks.org/introduction-to-linked-list-data-structure-and-algorithm-tutorial/)
3. [Applications, Advantages and Disadvantages of Linked List](https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-linked-list/)
4. [Linked List vs Array](https://www.geeksforgeeks.org/linked-list-vs-array/)

**Types of Linked List:**

1. [Introduction and Insertion in a Doubly Linked List](https://www.geeksforgeeks.org/introduction-and-insertion-in-a-doubly-linked-list/)
2. [Introduction to Circular Linked List](https://www.geeksforgeeks.org/circular-linked-list/)
3. [Circular Singly Linked List](https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/)
4. [Insertion in Doubly Circular Linked List](https://www.geeksforgeeks.org/insertion-in-doubly-circular-linked-list/)
5. [Types of Linked List](https://www.geeksforgeeks.org/types-of-linked-list/)

**Basic Operations:**

1. [Linked List Insertion](https://www.geeksforgeeks.org/linked-list-set-2-inserting-a-node/)
2. [Search an element in a Linked List (Iterative and Recursive)](https://www.geeksforgeeks.org/search-an-element-in-a-linked-list-iterative-and-recursive/)
3. [Find Length of a Linked List (Iterative and Recursive)](https://www.geeksforgeeks.org/find-length-of-a-linked-list-iterative-and-recursive/)
4. [Reverse a linked list](https://www.geeksforgeeks.org/write-a-function-to-reverse-the-nodes-of-a-linked-list/)
5. [Linked List Deletion (Deleting a given key)](https://www.geeksforgeeks.org/linked-list-set-3-deleting-node/)
6. [Linked List Deletion (Deleting a key at given position)](https://www.geeksforgeeks.org/delete-a-linked-list-node-at-a-given-position/)
7. [Write a function to delete a Linked List](https://www.geeksforgeeks.org/write-a-function-to-delete-a-linked-list/)
8. [Write a function to get Nth node in a Linked List](https://www.geeksforgeeks.org/write-a-function-to-get-nth-node-in-a-linked-list/)
9. [Nth node from the end of a Linked List](https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/)

**Standard problem on Linked List:**

* **Easy:**
  1. [Print the middle of a given linked list](https://www.geeksforgeeks.org/write-a-c-function-to-print-the-middle-of-the-linked-list/)
  2. [Write a function that counts the number of times a given int occurs in a Linked List](https://www.geeksforgeeks.org/write-a-function-that-counts-the-number-of-times-a-given-int-occurs-in-a-linked-list/)
  3. [Check if a linked list is Circular Linked List](https://www.geeksforgeeks.org/check-if-a-linked-list-is-circular-linked-list/)
  4. [Count nodes in Circular linked list](https://www.geeksforgeeks.org/count-nodes-circular-linked-list/)
  5. [Convert singly linked list into circular linked list](https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/)
  6. [Exchange first and last nodes in Circular Linked List](https://www.geeksforgeeks.org/exchange-first-last-node-circular-linked-list/)
  7. [Reverse a Doubly Linked List](https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/)
  8. [Program to find size of Doubly Linked List](https://www.geeksforgeeks.org/program-find-size-doubly-linked-list/)
  9. [An interesting method to print reverse of a linked list](https://www.geeksforgeeks.org/an-interesting-method-to-print-reverse-of-a-linked-list/)
  10. [Can we reverse a linked list in less than O(n)?](https://www.geeksforgeeks.org/can-we-reverse-a-linked-list-in-less-than-on/)
  11. [Circular Linked List Traversal](https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/)
  12. [Delete a node in a Doubly Linked List](https://www.geeksforgeeks.org/delete-a-node-in-a-doubly-linked-list/)
  13. [Reverse a Doubly Linked List](https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/)
* **Medium:**
  1. [Detect loop in a linked list](https://www.geeksforgeeks.org/write-a-c-function-to-detect-loop-in-a-linked-list/)
  2. [Find length of loop in linked list](https://www.geeksforgeeks.org/find-length-of-loop-in-linked-list/)
  3. [Remove duplicates from a sorted linked list](https://www.geeksforgeeks.org/remove-duplicates-from-a-sorted-linked-list/)
  4. [Intersection of two Sorted Linked Lists](https://www.geeksforgeeks.org/intersection-of-two-sorted-linked-lists/)
  5. [QuickSort on Singly Linked List](https://www.geeksforgeeks.org/quicksort-on-singly-linked-list/)
  6. [Split a Circular Linked List into two halves](https://www.geeksforgeeks.org/split-a-circular-linked-list-into-two-halves/)
  7. [Deletion from a Circular Linked List](https://www.geeksforgeeks.org/deletion-circular-linked-list/)
  8. [Convert singly linked list into circular linked list](https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/)
  9. [The Great Tree-List Recursion Problem.](https://www.geeksforgeeks.org/the-great-tree-list-recursion-problem/)
  10. [Merge Sort for Doubly Linked List](https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/)
  11. [Find pairs with given sum in doubly linked list](https://www.geeksforgeeks.org/find-pairs-given-sum-doubly-linked-list/)
  12. [Insert value in sorted way in a sorted doubly linked list](https://www.geeksforgeeks.org/insert-value-sorted-way-sorted-doubly-linked-list/)
  13. [Remove duplicates from an unsorted doubly linked list](https://www.geeksforgeeks.org/remove-duplicates-unsorted-doubly-linked-list/)
  14. [Rotate Doubly linked list by N nodes](https://www.geeksforgeeks.org/rotate-doubly-linked-list-n-nodes/)
  15. [Given only a pointer to a node to be deleted in a singly linked list, how do you delete it?](https://www.geeksforgeeks.org/given-only-a-pointer-to-a-node-to-be-deleted-in-a-singly-linked-list-how-do-you-delete-it/)
  16. [Modify contents of Linked List](https://www.geeksforgeeks.org/modify-contents-linked-list/)
* **Hard:**
  1. [Detect loop in a linked list](https://www.geeksforgeeks.org/write-a-c-function-to-detect-loop-in-a-linked-list/)
  2. [Intersection point of two Linked Lists.](https://www.geeksforgeeks.org/write-a-function-to-get-the-intersection-point-of-two-linked-lists/)
  3. [Circular Queue | Set 2 (Circular Linked List Implementation)](https://www.geeksforgeeks.org/circular-queue-set-2-circular-linked-list-implementation/)
  4. [Josephus Circle using circular linked list](https://www.geeksforgeeks.org/josephus-circle-using-circular-linked-list/)
  5. [The Great Tree-List Recursion Problem.](https://www.geeksforgeeks.org/the-great-tree-list-recursion-problem/)
  6. [Copy a linked list with next and arbit pointer](https://www.geeksforgeeks.org/a-linked-list-with-next-and-arbit-pointer/)
  7. [Convert a given Binary Tree to Doubly Linked List | Set](https://www.geeksforgeeks.org/convert-a-given-binary-tree-to-doubly-linked-list-set-4/)
  8. [Priority Queue using doubly linked list](https://www.geeksforgeeks.org/priority-queue-using-doubly-linked-list/)
  9. [Reverse a doubly linked list in groups of given size](https://www.geeksforgeeks.org/reverse-doubly-linked-list-groups-given-size/)
  10. [Reverse a stack without using extra space in O(n)](https://www.geeksforgeeks.org/reverse-stack-without-using-extra-space/)
  11. [Linked List representation of Disjoint Set Data Structures](https://www.geeksforgeeks.org/linked-list-representation-disjoint-set-data-structures/)
  12. [Sublist Search (Search a linked list in another list)](https://www.geeksforgeeks.org/sublist-search-search-a-linked-list-in-another-list/)
  13. [Construct a linked list from 2D matrix](https://www.geeksforgeeks.org/construct-linked-list-2d-matrix/)
  14. [Partitioning a linked list around a given value and If we don’t care about making the elements of the list “stable”](https://www.geeksforgeeks.org/partitioning-linked-list-around-given-value-dont-care-making-elements-list-stable/)

**Quick Links :**

* [‘Practice Problems’ on Linked List](https://practice.geeksforgeeks.org/topics/Linked-List/)
* [‘Videos’ on Linked List](https://www.youtube.com/playlist?list=PLqM7alHXFySH41ZxzrPNj2pAYPOI8ITe7)
* [‘Quizzes’ on Linked List](https://www.geeksforgeeks.org/data-structure-gq/linked-list-gq/)

**Easy questions:**

**Find the middle of a given linked list**

Given a singly linked list, find the middle of the linked list. For example, if the given linked list is 1->2->3->4->5 then the output should be 3.

If there are even nodes, then there would be two middle nodes, we need to print the second middle element. For example, if the given linked list is 1->2->3->4->5->6 then the output should be 4.

Recommended Problem

Finding middle element in a linked list

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+17 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/finding-middle-element-in-a-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 2.3L

**Method 1:** Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.

Below is the implementation of the above approach:

# Python program for the above approach

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** NodeOperation:

    # Function to add a new node

**def** pushNode(self, head\_ref, data\_val):

        # Allocate node and put in the data

        new\_node **=** Node(data\_val)

        # Link the old list of the new node

        new\_node.next **=** head\_ref

        # move the head to point to the new node

        head\_ref **=** new\_node

**return** head\_ref

    # A utility function to print a given linked list

**def** printNode(self, head):

**while** (head !**=** None):

            print('%d->' **%** head.data, end**=**"")

            head **=** head.next

**print**("NULL")

    ''' Utility Function to find length of linked list '''

**def** getLen(self, head):

        temp **=** head

        len **=** 0

**while** (temp !**=** None):

            len **+=** 1

            temp **=** temp.next

**return** len

**def** printMiddle(self, head):

**if** head !**=** None:

            # find length

            len **=** self.getLen(head)

            temp **=** head

            # traverse till we reached half of length

            midIdx **=** len **//** 2

**while** midIdx !**=** 0:

                temp **=** temp.next

                midIdx **-=** 1

            # temp will be storing middle element

            print('The middle element is [%d]' **%** temp.data)

# Driver Code

head **=** None

temp **=** NodeOperation()

**for** i **in** range(5, 0, **-**1):

    head **=** temp.pushNode(head, i)

    temp.printNode(head)

    temp.printMiddle(head)

# This code is contributed by Tapesh(tapeshdua420)

**Output**

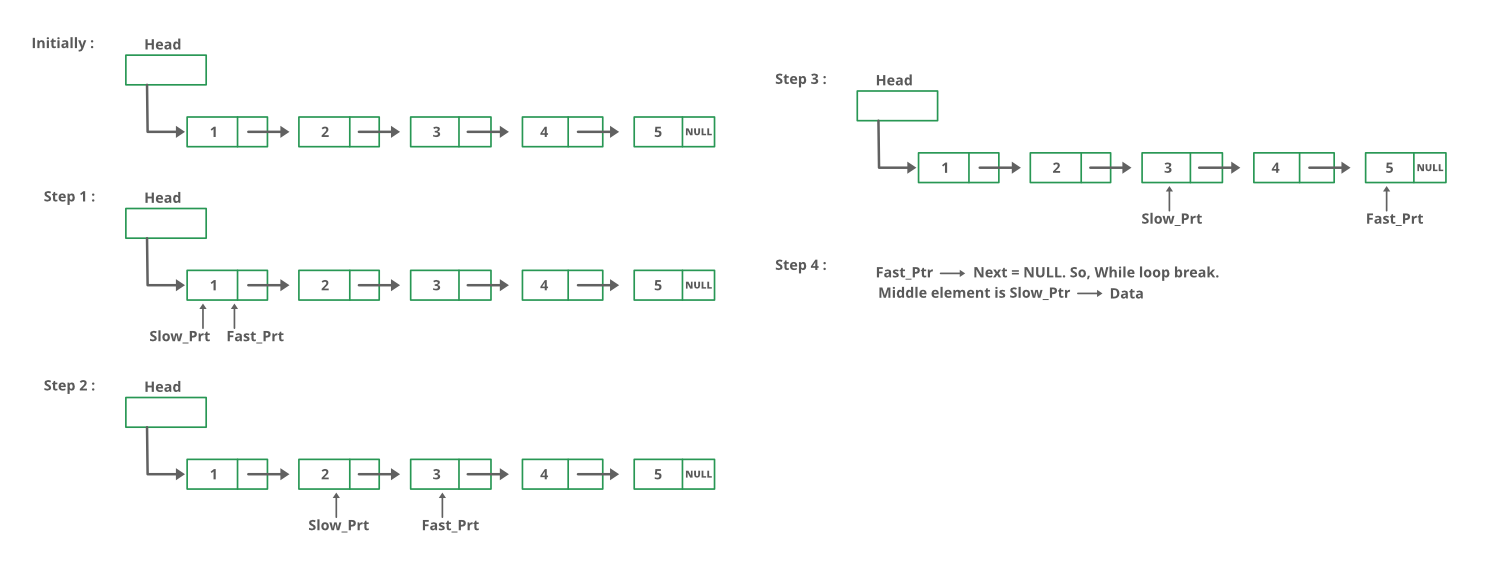
5->NULL  
The middle element is [5]  
4->5->NULL  
The middle element is [5]  
3->4->5->NULL  
The middle element is [4]  
2->3->4->5->NULL  
The middle element is [4]  
1->2->3->4->5->NULL  
The middle element is [3]

**Time Complexity**: O(n) where n is no of nodes in linked list

**Auxiliary Space:**O(1)

**Method 2:** Traverse linked list using two-pointers. Move one pointer by one and the other pointers by two. When the fast pointer reaches the end, the slow pointer will reach the middle of the linked list.

Below image shows how printMiddle function works in the code :



# Python3 program to find middle of linked list

# Node class

**class** Node:

    # Function to initialise the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data  # Assign data

        self.next **=** None  # Initialize next as null

# Linked List class contains a Node object

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new node at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Print the linked list

**def** printList(self):

        node **=** self.head

**while** node:

            print(str(node.data) **+** "->", end**=**"")

            node **=** node.next

        print("NULL")

    # Function that returns middle.

**def** printMiddle(self):

        # Initialize two pointers, one will go one step a time (slow), another two at a time (fast)

        slow **=** self.head

        fast **=** self.head

        # Iterate till fast's next is null (fast reaches end)

**while** fast **and** fast.next:

            slow **=** slow.next

            fast **=** fast.next.next

        # return the slow's data, which would be the middle element.

**print**("The middle element is ", slow.data)

# Code execution starts here

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Start with the empty list

    llist **=** LinkedList()

**for** i **in** range(5, 0, **-**1):

        llist.push(i)

        llist.printList()

        llist.printMiddle()

 # Code is contributed by Kumar Shivam (kshivi99)

**Output**

5->NULL  
The middle element is [5]  
4->5->NULL  
The middle element is [5]  
3->4->5->NULL  
The middle element is [4]  
2->3->4->5->NULL  
The middle element is [4]  
1->2->3->4->5->NULL  
The middle element is [3]

**Time Complexity:** O(N), As we are traversing the list only once.

**Auxiliary Space:** O(1), As constant extra space is used.

**Method 3:** Initialize the mid element as head and initialize a counter as 0. Traverse the list from the head, while traversing increment the counter and change mid to mid->next whenever the counter is odd. So the mid will move only half of the total length of the list.

Thanks to Narendra Kangralkar for suggesting this method.

# Node class

**class** Node:

    # Function to initialise the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data  # Assign data

        self.next **=** None  # Initialize next as null

# Linked List class contains a Node object

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new node at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Print the linked list

**def** printList(self):

        node **=** self.head

**while** node:

            print(str(node.data) **+** "->", end **=** "")

            node **=** node.next

        print("NULL")

    # Function to get the middle of

    #  the linked list

**def** printMiddle(self):

        count **=** 0

        mid **=** self.head

        heads **=** self.head

**while**(heads !**=** None):

        # Update mid, when 'count'

        # is odd number

**if** count&1:

                mid **=** mid.next

            count **+=** 1

            heads **=** heads.next

        # If empty list is provided

**if** mid!**=**None:

**print**("The middle element is ", mid.data)

# Code execution starts here

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Start with the empty list

    llist **=** LinkedList()

**for** i **in** range(5, 0, **-**1):

        llist.push(i)

        llist.printList()

        llist.printMiddle()

 # This Code is contributed by Manisha\_Ediga

**Output**

5->NULL  
The middle element is [5]

4->5->NULL  
The middle element is [5]

3->4->5->NULL  
The middle element is [4]

2->3->4->5->NULL  
The middle element is [4]

1->2->3->4->5->NULL  
The middle element is [3]

**Time Complexity:**O(N), As we are traversing the list once.

**Auxiliary Space:**O(1), As constant extra space is used.

*From <*[*https://www.geeksforgeeks.org/write-a-c-function-to-print-the-middle-of-the-linked-list/*](https://www.geeksforgeeks.org/write-a-c-function-to-print-the-middle-of-the-linked-list/)*>*

**Write a function that counts the number of times a given int occurs in a Linked List**

Given a singly linked list and a key, count the number of occurrences of the given key in the linked list. For example, if the given linked list is 1->2->1->2->1->3->1 and the given key is 1, then the output should be 4.

Recommended Problem

Occurence of an integer in a Linked List

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/occurence-of-an-integer-in-a-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 49K

**Method 1- Without Recursion**

**Algorithm:**

Step 1: Start  
Step 2: Create A Function Of A Linked List, Pass A Number   
 As Arguments And Provide The Count Of The Number To The Function.  
Step 3: Initialize Count Equal To 0.  
Step 4: Traverse In Linked List Until Equal Number Found.  
Step 5: If Found A Number Equal To Update Count By 1.  
Step 6: After Reaching The End Of The Linked List Return Count.  
Step 7: Call The Function.  
Step 8: Prints The Number Of Int Occurrences.  
Step 9: Stop.

**Implementation:**

# Python program to count the number of time a given

# int occurs in a linked list

# Node class

**class** Node:

    # Constructor to initialize the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Counts the no . of occurrences of a node

    # (search\_for) in a linked list (head)

**def** count(self, search\_for):

        current **=** self.head

        count **=** 0

**while**(current **is not** None):

**if** current.data **==** search\_for:

                count **+=** 1

            current **=** current.next

**return** count

    # Function to insert a new node at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Utility function to print the LinkedList

**def** printList(self):

        temp **=** self.head

**while**(temp):

**print** (temp.data)

            temp **=** temp.next

# Driver program

llist **=** LinkedList()

llist.push(1)

llist.push(3)

llist.push(1)

llist.push(2)

llist.push(1)

# Check for the count function

print ("count of 1 is % d" **%**(llist.count(1)))

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

count of 1 is 3

**Time Complexity:** O(n)

**Auxiliary Space:** O(1)

**Method 2- With Recursion**

**Algorithm:**

**Algorithm**  
count(head, key);  
if head is NULL  
return frequency  
if(head->data==key)  
increase frequency by 1  
count(head->next, key)

**Implementation:**

# Python program to count the number of

# time a given int occurs in a linked list

# Node class

**class** Node:

    # Constructor to initialize the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

        self.counter **=** 0

    # Counts the no . of occurrences of a node

    # (seach\_for) in a linked list (head)

**def** count(self, li, key):

        # Base case

**if**(**not** li):

**return** self.counter

        # If key is present in

        # current node, return true

**if**(li.data **==** key):

            self.counter **=** self.counter **+** 1

        # Recur for remaining list

**return** self.count(li.next, key)

    # Function to insert a new node

    # at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Utility function to print the

    # linked LinkedList

**def** printList(self):

        temp **=** self.head

**while**(temp):

**print** (temp.data)

            temp **=** temp.next

# Driver Code

llist **=** LinkedList()

llist.push(1)

llist.push(3)

llist.push(1)

llist.push(2)

llist.push(1)

# Check for the count function

print ("count of 1 is", llist.count(llist.head, 1))

# This code is contributed by

# Gaurav Kumar Raghav

**Output**

count of 1 is 3

**Time complexity**: O(n) where n is size of linked list

**Auxiliary Space**: O(n) for call stack since using recursion

Below method can be used to avoid Global variable ‘frequency'(counter in case of Python 3 Code).

**def** count(self, temp, key):

    # during the initial call, temp

    # has the value of head

    # Base case

**if** temp **is** None:

**return** 0

    # if a match is found, we

    # increment the counter

**if** temp.data **==** key:

**return** 1 **+** count(temp.next, key)

**return** count(temp.next, key)

# to call count, use

# linked\_list\_name.count(head, key)

The above method implements head recursion. Below given is the tail recursive implementation for the same. Thanks to **Puneet Jain**for suggesting this approach:

int count(struct Node\* head, int key)  
{  
 if(head == NULL)  
 return 0;  
   
 int frequency = count(head->next, key);  
 if(head->data == key)  
 return 1 + frequency;  
   
 // else   
 return frequency;  
}

**Time Complexity :** O(n)

**Auxiliary Space :**O(n)

*From <*[*https://www.geeksforgeeks.org/write-a-function-that-counts-the-number-of-times-a-given-int-occurs-in-a-linked-list/*](https://www.geeksforgeeks.org/write-a-function-that-counts-the-number-of-times-a-given-int-occurs-in-a-linked-list/)*>*

**Check if a linked list is Circular Linked List**

Given a singly linked list, find if the linked list is [circular](https://www.geeksforgeeks.org/circular-linked-list/) or not.

*A linked list is called circular if it is not NULL-terminated and all nodes are connected in the form of a cycle. Below is an example of a circular linked list.*

[](https://media.geeksforgeeks.org/wp-content/uploads/Circular-Linked-List-Diagram.png)

**Note:**

* An empty linked list is considered circular.
* This problem is different from [cycle detection problem](https://practice.geeksforgeeks.org/problems/detect-loop-in-linked-list/1), here all nodes have to be part of cycle.

Recommended Problem

Check If Circular Linked List

[circular-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=circular-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+1 more

[MAQ Software](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=MAQ%20Software&sortBy=submissions)

[Microsoft](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Microsoft&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/circular-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.1L

*The idea is to store head of the linked list and traverse it. If iterator reaches NULL, linked list is not circular. else If it reaches head again, then linked list is circular.*

Follow the given steps to solve the problem:

* Declare a Node pointer node and initialize it to the head’s next
* Move node pointer to the next node, while the node is not equal to nullptr and node is not equal to the head
* After coming out of the loop, check if the node is equal to head then return true, else return false

Below is the Implementation of the above approach:

# Python3 program to check if a linked list is circular

# Node class

**class** Node:

    # Function to initialise the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data  # Assign data

        self.next **=** None  # Initialize next as null

# Linked List class contains a Node object

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

**def** Circular(head):

**if** head **==** None:

**return** True

    # Next of head

    node **=** head.next

    i **=** 0

    # This loop would stop in both cases (1) If

    # Circular (2) Not circular

**while**((node **is not** None) **and** (node **is not** head)):

        i **=** i **+** 1

        node **=** node.next

**return**(node **==** head)

# Driver's code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    llist **=** LinkedList()

    llist.head **=** Node(1)

    second **=** Node(2)

    third **=** Node(3)

    fourth **=** Node(4)

    llist.head.next **=** second

    second.next **=** third

    third.next **=** fourth

**if** (Circular(llist.head)):

**print**('Yes')

**else**:

**print**('No')

    fourth.next **=** llist.head

**if** (Circular(llist.head)):

        print('Yes')

**else**:

**print**('No')

# This code is contributed by Sanket Badhe

**Output**

No  
Yes

**Time Complexity:** O(N)

**Auxiliary Space:** O(1)

*From <*[*https://www.geeksforgeeks.org/check-if-a-linked-list-is-circular-linked-list/*](https://www.geeksforgeeks.org/check-if-a-linked-list-is-circular-linked-list/)*>*

**Count nodes in Circular linked list**

Given a circular linked list, count the number of nodes in it. For example, the output is 5 for the below list.

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image004.png

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:**We use the concept used in [Circular Linked List | Set 2 (Traversal)](https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/). While traversing, we keep track of the count of nodes.

 Below is the implementation of the above idea:

# Python3 program to count number of nodes in

# a circular linked list.

# structure for a node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# Function to insert a node at the beginning

# of a Circular linked list \*/

**def** push(head\_ref,data):

    ptr1 **=** Node(0)

    temp **=** head\_ref

    ptr1.data **=** data

    ptr1.next **=** head\_ref

    # If the linked list is not None then set

    # the next of last node

**if** (head\_ref !**=** None) :

**while** (temp.next !**=** head\_ref):

            temp **=** temp.next

        temp.next **=** ptr1

**else**:

        ptr1.next **=** ptr1 #For the first node \*/

    head\_ref **=** ptr1

**return** head\_ref

# Function to print nodes

# in a given Circular linked list

**def** countNodes(head):

    temp **=** head

    result **=** 0

**if** (head !**=** None) :

**while** True :

            temp **=** temp.next

            result **=** result **+** 1

**if** (temp **==** head):

**break**

**return** result

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Initialize lists as empty \*/

    head **=** None

    head **=** push(head, 12)

    head **=** push(head, 56)

    head **=** push(head, 2)

    head **=** push(head, 11)

**print**( countNodes(head))

# This code is contributed by Arnab Kundu

**Output**

4

**Time Complexity:** **O(n),** As we are visiting every node just once.

**Auxiliary Space: O(1),**As constant extra space is used

**Count nodes in Circular linked list**

Given a circular linked list, count the number of nodes in it. For example, the output is 5 for the below list.

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image004.png

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:**We use the concept used in [Circular Linked List | Set 2 (Traversal)](https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/). While traversing, we keep track of the count of nodes.

 Below is the implementation of the above idea:

# Python3 program to count number of nodes in

# a circular linked list.

# structure for a node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# Function to insert a node at the beginning

# of a Circular linked list \*/

**def** push(head\_ref,data):

    ptr1 **=** Node(0)

    temp **=** head\_ref

    ptr1.data **=** data

    ptr1.next **=** head\_ref

    # If the linked list is not None then set

    # the next of last node

**if** (head\_ref !**=** None) :

**while** (temp.next !**=** head\_ref):

            temp **=** temp.next

        temp.next **=** ptr1

**else**:

        ptr1.next **=** ptr1 #For the first node \*/

    head\_ref **=** ptr1

**return** head\_ref

# Function to print nodes

# in a given Circular linked list

**def** countNodes(head):

    temp **=** head

    result **=** 0

**if** (head !**=** None) :

**while** True :

            temp **=** temp.next

            result **=** result **+** 1

**if** (temp **==** head):

**break**

**return** result

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Initialize lists as empty \*/

    head **=** None

    head **=** push(head, 12)

    head **=** push(head, 56)

    head **=** push(head, 2)

    head **=** push(head, 11)

**print**( countNodes(head))

# This code is contributed by Arnab Kundu

**Output**

4

**Time Complexity:** **O(n),** As we are visiting every node just once.

**Auxiliary Space: O(1),**As constant extra space is used

*From <*[*https://www.geeksforgeeks.org/count-nodes-circular-linked-list/*](https://www.geeksforgeeks.org/count-nodes-circular-linked-list/)*>*

**Convert singly linked list into circular linked list**

Given a singly linked list, we have to convert it into circular linked list. For example, we have been given a singly linked list with four nodes and we want to convert this singly linked list into circular linked list.



The above singly linked list is converted into circular linked list.



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:** The idea is to traverse the singly linked list and check if the node is the last node or not. If the node is the last node i.e pointing to NULL then make it point to the starting node i.e head node. Below is the implementation of this approach.

**Implementation:**

# Python3 program for converting

# singly linked list into

# circular linked list.

**import** sys

# Linked list node

**class** Node:

**def** \_\_init\_\_(self,data):

        self.data **=** data

        self.next **=** None

**def** push(head, data):

**if not** head:

**return** Node(data)

    # Allocate dynamic memory

    # for newNode.

    # Assign the data into newNode.

    newNode **=** Node(data)

    # newNode.next assign the

    # address of head node.

    newNode.next **=** head

    # newNode become the headNode.

    head **=** newNode

**return** head

# Function that convert

# singly linked list into

# circular linked list.

**def** circular(head):

    # declare a node variable

    # start and assign head

    # node into start node.

    start **=** head

    # check that while head.next

    # not equal to null then head

    # points to next node.

**while**(head.next **is not** None):

        head **=** head.next

    # if head.next points to null

    # then start assign to the

    # head.next node.

    head.next **=** start

**return** start

# Function that display the elements

# of circular linked list.

**def** displayList(node):

    start **=** node

**while**(node.next **is not** start):

**print**("{} ".format(node.data),end**=**"")

        node**=**node.next

    # Display the last node of

    # circular linked list.

    print("{} ".format(node.data),end**=**"")

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Start with empty list

    head**=**None

    # Using push() function to

    # convert singly linked list

    # 17.22.13.14.15

    head**=**push(head,15)

    head**=**push(head,14)

    head**=**push(head,13)

    head**=**push(head,22)

    head**=**push(head,17)

    # Call the circular\_list function

    # that convert singly linked

    # list to circular linked list.

    circular(head)

    print("Display List:")

    displayList(head)

# This Code is Contributed By Vikash Kumar 37

**Output**

Display list:   
17 22 13 14 15

**Time Complexity:** **O(n),**As we need to move through the whole list to get hold of the last node.

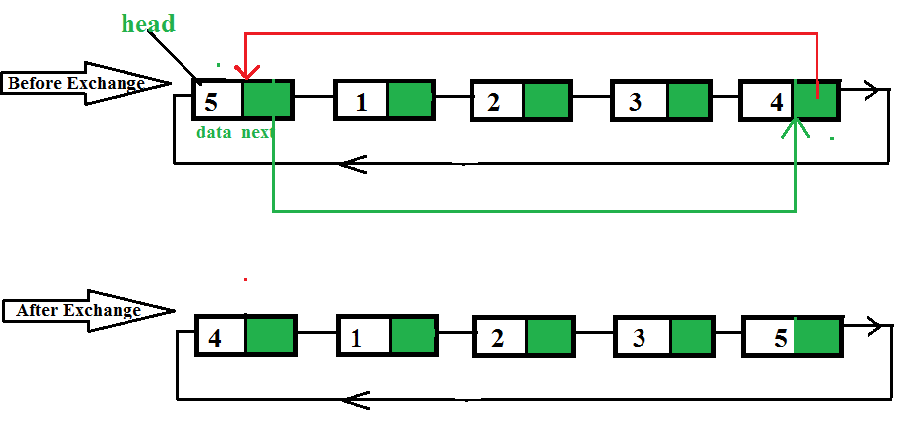
**Auxiliary Space: O(1),**As constant extra space is used.

*From <*[*https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/*](https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/)*>*

**Exchange first and last nodes in Circular Linked List**

Given [Circular linked list](https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/) exchange the first and the last node. The task should be done with only one extra node, you can not declare more than one extra node, and also you are not allowed to declare any other temporary variable.

**Note:** Extra node means the need of a node to traverse a list.



**Examples:**

**Input :** 5 4 3 2 1  
**Output :** 1 4 3 2 5

**Input :** 6 1 2 3 4 5 6 7 8 9  
**Output :** 9 1 2 3 4 5 6 7 8 6

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1:***(By Changing Links of First and Last Nodes)*

We first find a pointer to the previous to the last node. Let this node be p. Now we change the next links so that the last and first nodes are swapped.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to exchange first and

# last node in circular linked list

**import** math

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**def** addToEmpty(head, data):

    # This function is only for empty list

**if** (head !**=** None):

**return** head

    # Creating a node dynamically.

    temp **=** Node(data)

    # Assigning the data.

    temp.data **=** data

    head **=** temp

    # Creating the link.

    head.next **=** head

**return** head

**def** addBegin(head, data):

**if** (head **==** None):

**return** addToEmpty(head, data)

    temp **=** Node(data)

    temp.data **=** data

    temp.next **=** head.next

    head.next **=** temp

**return** head

# function for traversing the list

**def** traverse(head):

    # If list is empty, return.

**if** (head **==** None):

        print("List is empty.")

**return**

    # Pointing to first Node of the list.

    p **=** head

**print**(p.data, end**=**" ")

    p **=** p.next

    # Traversing the list.

**while**(p !**=** head):

**print**(p.data, end**=**" ")

        p **=** p.next

**def** exchangeNodes(head):

    # Cases Handled: Linked List either empty or containing single node.

**if** head **==** None **or** head.next **==** head:

**return** head

    # Cases Handled: Linked List containing only two nodes

**elif** head.next.next **==** head:

        head **=** head.next

**return** head

    # Cases Handled: Linked List containing multiple nodes

**else**:

        prev **=** None

        curr **=** head

        temp **=** head

        # finding last and second last nodes in linkedlist list

**while** curr.next !**=** head:

            prev **=** curr

            curr **=** curr.next

        # point the last node to second node of the list

        curr.next **=** temp.next

        # point the second last node to first node

        prev.next **=** temp

        # point the end of node to start ( make linked list circular )

        temp.next **=** curr

        # mark the starting of linked list

        head **=** curr

**return** head

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    head **=** None

    head **=** addToEmpty(head, 6)

**for** x **in** range(5, 0, **-**1):

        head **=** addBegin(head, x)

**print**("List Before: ", end**=**"")

    traverse(head)

**print**()

**print**("List After: ", end**=**"")

    head **=** exchangeNodes(head)

    traverse(head)

# This code is contributed by Srathore

# Improved by Vinay Kumar (vinaykumar71)

**Output**

List Before: 6 1 2 3 4 5   
List After: 5 1 2 3 4 6

*From <*[*https://www.geeksforgeeks.org/exchange-first-last-node-circular-linked-list/*](https://www.geeksforgeeks.org/exchange-first-last-node-circular-linked-list/)*>*

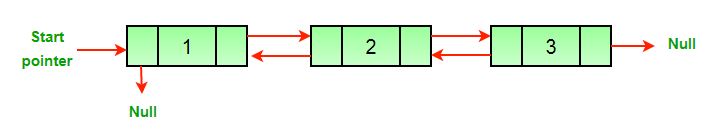
**Reverse a Doubly Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 10 Nov, 2022
* Read
* Discuss(99)
* Courses
* Practice
* Video

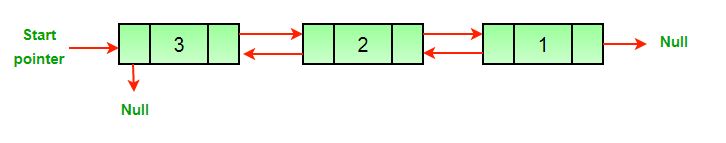
Given a [Doubly Linked List](https://www.geeksforgeeks.org/doubly-linked-list/), the task is to reverse the given Doubly Linked List.

**Example:**

***Input:***



***Output:***



[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Recommended Problem

Reverse a Doubly Linked List

[doubly-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=doubly-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+1 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[D-E-Shaw](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=D-E-Shaw&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/reverse-a-doubly-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 57.3K

Follow the given steps to solve the problem using the above approach:

* Traverse the linked list using a pointer
* Swap the prev and next pointers for all nodes
* At last, change the head pointer of the doubly linked list

Below is the implementation of the above approach:

* C
* C++
* Java
* Python3
* C#
* Javascript

# Python3 Program to reverse a doubly linked list

# A node of the doubly linked list

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

     # Constructor for empty Doubly Linked List

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function reverse a Doubly Linked List

**def** reverse(self):

        temp **=** None

        current **=** self.head

        # Swap next and prev for all nodes of

        # doubly linked list

**while** current **is not** None:

            temp **=** current.prev

            current.prev **=** current.next

            current.next **=** temp

            current **=** current.prev

        # Before changing head, check for the cases like

        # empty list and list with only one node

**if** temp **is not** None:

            self.head **=** temp.prev

    # Given a reference to the head of a list and an

    # integer,inserts a new node on the front of list

**def** push(self, new\_data):

        # 1. Allocates node

        # 2. Put the data in it

        new\_node **=** Node(new\_data)

        # 3. Make next of new node as head and

        # previous as None (already None)

        new\_node.next **=** self.head

        # 4. change prev of head node to new\_node

**if** self.head **is not** None:

            self.head.prev **=** new\_node

        # 5. move the head to point to the new node

        self.head **=** new\_node

**def** printList(self, node):

**while**(node **is not** None):

            print(node.data, end**=**' ')

            node **=** node.next

# Driver's code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    dll **=** DoublyLinkedList()

    dll.push(2)

    dll.push(4)

    dll.push(8)

    dll.push(10)

**print**("\nOriginal Linked List")

    dll.printList(dll.head)

    # Function call

    dll.reverse()

    print("\nReversed Linked List")

    dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Original Linked list  
10 8 4 2   
Reversed Linked list  
2 4 8 10

**Time Complexity:**O(N), where N denotes the number of nodes in the doubly linked list.

**Auxiliary Space:**O(1)

*From <*[*https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/*](https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/)*>*

**Program to find size of Doubly Linked List**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 12 Aug, 2022
* Read
* Discuss(1)
* Courses
* Practice
* Video

Given a [doubly linked list](https://www.geeksforgeeks.org/doubly-linked-list/), the task is to find the size of that doubly linked list. For example, the size of the below linked list is 4.



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes.

Traversal of a doubly linked list can be in either direction. In fact, the direction of traversal can change many times, if desired.

For example, the function should return 3 for the above doubly linked list.

**Algorithm :**

1. *Initialize size to 0.*
2. *Initialize a node pointer, temp = head.*
3. *Do following while temp is not NULL*
4. *temp = temp -> next*
5. *size++;*
6. *Return size.*

**Implementation:**

* C++
* C
* Java
* Python3
* C#
* Javascript

# A complete working Python3 program to

# find size of doubly linked list.

# A linked list node

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** None

        self.next **=** None

        self.prev **=** None

# Function to add a node to front of doubly

# linked list

**def** push( head\_ref, new\_data):

    new\_node **=** Node()

    new\_node.data **=** new\_data

    new\_node.next **=** (head\_ref)

    new\_node.prev **=** None

**if** ((head\_ref) !**=** None):

        (head\_ref).prev **=** new\_node

    (head\_ref) **=** new\_node

**return** head\_ref

# This function returns size of linked list

**def** findSize(node):

    res **=** 0

**while** (node !**=** None):

        res **=** res **+** 1

        node **=** node.next

**return** res

# Driver code

head **=** None

head **=** push(head, 4)

head **=** push(head, 3)

head **=** push(head, 2)

head **=** push(head, 1)

**print**(findSize(head))

# This code is contributed by Arnab Kundu

**Output:**

4

*From <*[*https://www.geeksforgeeks.org/program-find-size-doubly-linked-list/*](https://www.geeksforgeeks.org/program-find-size-doubly-linked-list/)*>*

**An interesting method to print reverse of a linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 19 Dec, 2022
* Read
* Discuss(2)
* Courses
* Practice
* Video

We are given a linked list, we need to print the linked list in reverse order.

Examples:

Input : list : 5-> 15-> 20-> 25   
Output : Reversed Linked list : 25-> 20-> 15-> 5

Input : list : 85-> 15-> 4-> 20   
Output : Reversed Linked list : 20-> 4-> 15-> 85

Input : list : 85  
Output : Reversed Linked list : 85

For printing a list in reverse order, we have already discussed [Iterative and Recursive Methods to Reverse](https://www.geeksforgeeks.org/write-a-function-to-reverse-the-nodes-of-a-linked-list/).

In this post, an interesting method is discussed, that doesn’t require recursion and does no modifications to list. The function also visits every node of linked list only once.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Trick :** Idea behind printing a list in reverse order without any recursive function or loop is to use [Carriage return](https://en.wikipedia.org/wiki/Carriage_return) (“r”). For this, we should have knowledge of length of list. Now, we should print n-1 blank space and then print 1st element then “r”, further again n-2 blank space and 2nd node then “r” and so on..

**Carriage return (“r”) :** It commands a printer (cursor or the display of a system console), to move the position of the cursor to the first position on the same line.

C/C++

* Java
* C#
* Python3

# Python3 program to print reverse of list

# Link list node

**class** Node:

**def** \_\_init\_\_(self):

        self.data**=**  0

        self.next**=**None

# Function to reverse the linked list

**def** printReverse( head\_ref, n):

    j **=** 0

    current **=** head\_ref

**while** (current !**=** None):

        i **=** 0

        # For each node, print proper number

        # of spaces before printing it

**while** ( i < 2 **\*** (n **-** j) ):

**print**(end**=**" ")

            i **=** i **+** 1

        # use of carriage return to move back

        # and print.

        print( current.data, end **=** "\r")

        current **=** current.next

        j **=** j **+** 1

 # Function to push a node

**def** push( head\_ref, new\_data):

    new\_node **=** Node()

    new\_node.data **=** new\_data

    new\_node.next **=** (head\_ref)

    (head\_ref) **=** new\_node

**return** head\_ref;

# Function to print linked list and find its

#  length

**def** printList( head):

    # i for finding length of list

    i **=** 0

    temp **=** head

**while** (temp !**=** None):

**print**( temp.data,end **=** " ")

        temp **=** temp.next

        i **=** i **+** 1

**return** i

# Driver program to test above function

# Start with the empty list

head **=** None

# list nodes are as 6 5 4 3 2 1

head **=** push(head, 1)

head **=** push(head, 2)

head **=** push(head, 3)

head **=** push(head, 4)

head **=** push(head, 5)

head **=** push(head, 6)

print("Given linked list:")

# printlist print the list and

# return the size of list

n **=** printList(head)

# print reverse list with help

# of carriage return function

**print**("\nReversed Linked list:")

printReverse(head, n)

print()

# This code is contributed by Arnab Kundu

**Output:**

Given linked list:  
6 5 4 3 2 1  
Reversed Linked List:  
1 2 3 4 5 6

**Time Complexity: O(N).**

**Auxiliary Space: O(N),** where, N is the total number of nodes.

**Input and Output Illustration :**

**Input: 6 5 4 3 2 1**

**1st Iteration \_ \_ \_ \_ \_ 6**

**2nd Iteration \_ \_ \_ \_ 5 6**

**3rd Iteration \_ \_ \_ 4 5 6**

**4th Iteration \_ \_ 3 4 5 6**

**5th Iteration \_ 2 3 4 5 6**

**Final Output 1 2 3 4 5 6**

**NOTE:**Above program may not work on online compilers because they do not support anything like carriage return on their console.

*From <*[*https://www.geeksforgeeks.org/an-interesting-method-to-print-reverse-of-a-linked-list/*](https://www.geeksforgeeks.org/an-interesting-method-to-print-reverse-of-a-linked-list/)*>*

**Traversal of Circular Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 14 Nov, 2022
* Read
* Discuss(10)
* Courses
* Practice
* Video

We have discussed [Circular Linked List Introduction and Applications,](https://www.geeksforgeeks.org/circular-linked-list/)in the previous post on Circular Linked List. In this post, traversal operation is discussed.

C:\Users\qj771f\AppData\Local\Temp\msohtmlclip1\02\clip_image011.png

Recommended Problem

Circular Linked List Traversal

[circular-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=circular-linked-list&sortBy=submissions)

[Cisco](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Cisco&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/circular-linked-list-traversal/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 18.1K

In a conventional linked list, we traverse the list from the head node and stop the traversal when we reach NULL. In a circular linked list, we stop traversal when we reach the first node again. Following is the C code for the linked list traversal.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Function to print nodes in a given Circular linked list

**def** printList(self):

    temp **=** self.head

    # If linked list is not empty

**if** self.head **is not** None:

**while**(True):

            # Print nodes till we reach first node again

            print(temp.data, end**=**" ")

            temp **=** temp.next

**if** (temp **==** self.head):

**break**

**Time Complexity:** O(n)

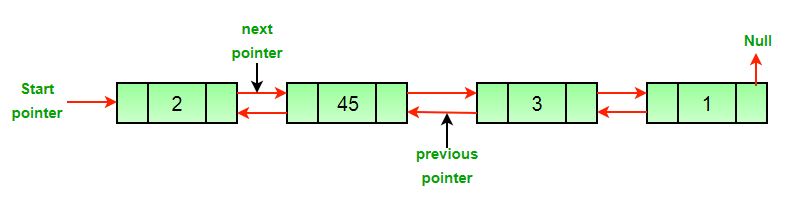
**Auxiliary Space:** O(1)

*From <*[*https://www.geeksforgeeks.org/traversal-of-circular-linked-list/*](https://www.geeksforgeeks.org/traversal-of-circular-linked-list/)*>*

**Delete a node in a Doubly Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 21 Nov, 2022
* Read
* Discuss(75)
* Courses
* Practice
* Video

Write a function to delete a given node in a [doubly-linked list](https://www.geeksforgeeks.org/doubly-linked-list/).



**Example:**

***Input:****DLL = 2->45->3->1, Node = 45*

***Output:****2->3->1*

***Input:****DLL = 2->45->3->1, Node = 1*

***Output:****2->45->3*

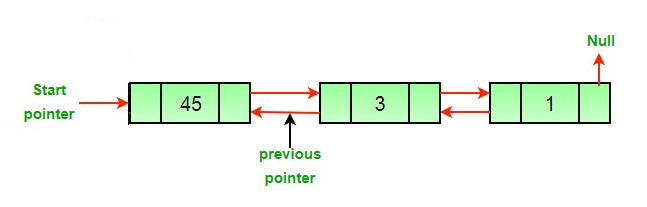
Recommended Practice

[Delete node in Doubly Linked List](https://practice.geeksforgeeks.org/problems/delete-node-in-doubly-linked-list/1/)

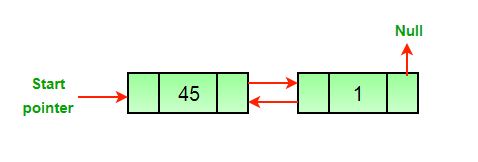
[Try It!](https://practice.geeksforgeeks.org/problems/delete-node-in-doubly-linked-list/1/)

**Approach:** The deletion of a node in a doubly-linked list can be divided into three main categories:

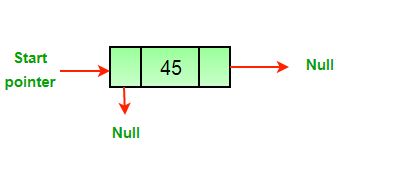
* **After the deletion of the head node.**



* **After the deletion of the middle node.**

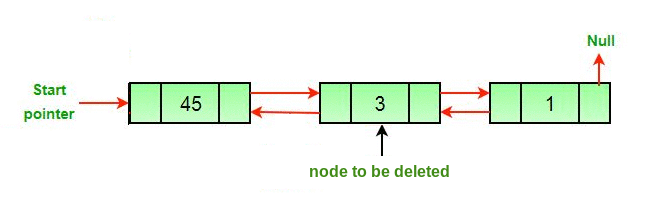


* **After the deletion of the last node.**



**All three mentioned cases can be handled in two steps if the pointer of the node to be deleted and the head pointer is known.**

1. If the node to be deleted is the head node then make the next node as head.
2. If a node is deleted, connect the next and previous node of the deleted node.



[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Algorithm:**

* Let the node to be deleted be *del*.
* If node to be deleted is head node, then change the head pointer to next current head.

if *headnode* == *del* then  
 *headnode* = *del*.nextNode

* Set prev of next to del, if next to del exists.

if *del*.nextNode != *none*   
 *del*.nextNode.previousNode = *del*.previousNode

* Set next of previous to del, if previous to del exists.

if *del*.previousNode != *none*   
 *del*.previousNode.nextNode = *del*.next

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Program to delete a node in a doubly-linked list

# for Garbage collection

**import** gc

# A node of the doubly linked list

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

     # Constructor for empty Doubly Linked List

**def** \_\_init\_\_(self):

        self.head **=** None

   # Function to delete a node in a Doubly Linked List.

   # head\_ref --> pointer to head node pointer.

   # dele --> pointer to node to be deleted

**def** deleteNode(self, dele):

        # Base Case

**if** self.head **is** None **or** dele **is** None:

**return**

        # If node to be deleted is head node

**if** self.head **==** dele:

            self.head **=** dele.next

        # Change next only if node to be deleted is NOT

        # the last node

**if** dele.next **is not** None:

            dele.next.prev **=** dele.prev

        # Change prev only if node to be deleted is NOT

        # the first node

**if** dele.prev **is not** None:

            dele.prev.next **=** dele.next

        # Finally, free the memory occupied by dele

        # Call python garbage collector

        gc.collect()

    # Given a reference to the head of a list and an

    # integer, inserts a new node on the front of list

**def** push(self, new\_data):

        # 1. Allocates node

        # 2. Put the data in it

        new\_node **=** Node(new\_data)

        # 3. Make next of new node as head and

        # previous as None (already None)

        new\_node.next **=** self.head

        # 4. change prev of head node to new\_node

**if** self.head **is not** None:

            self.head.prev **=** new\_node

        # 5. move the head to point to the new node

        self.head **=** new\_node

**def** printList(self, node):

**while**(node **is not** None):

**print**(node.data,end**=**' ')

            node **=** node.next

# Driver program to test the above functions

# Start with empty list

dll **=** DoublyLinkedList()

# Let us create the doubly linked list 10<->8<->4<->2

dll.push(2);

dll.push(4);

dll.push(8);

dll.push(10);

print ("\n Original Linked List",end**=**' ')

dll.printList(dll.head)

# delete nodes from doubly linked list

dll.deleteNode(dll.head)

dll.deleteNode(dll.head.next)

dll.deleteNode(dll.head.next)

# Modified linked list will be NULL<-8->NULL

print("\n Modified Linked List",end**=**' ')

dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Original Linked list 10 8 4 2   
Modified Linked list 8

**Complexity Analysis:**

* **Time Complexity:** O(1).   
  Since traversal of the linked list is not required so the time complexity is constant.
* **Auxiliary Space:** O(1).   
  As no extra space is required, so the space complexity is constant.

*From <*[*https://www.geeksforgeeks.org/delete-a-node-in-a-doubly-linked-list/*](https://www.geeksforgeeks.org/delete-a-node-in-a-doubly-linked-list/)*>*

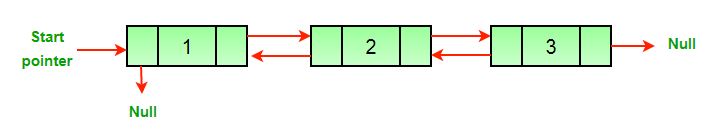
**Reverse a Doubly Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 10 Nov, 2022
* Read
* Discuss(99)
* Courses
* Practice
* Video

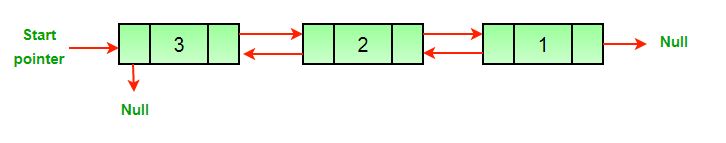
Given a [Doubly Linked List](https://www.geeksforgeeks.org/doubly-linked-list/), the task is to reverse the given Doubly Linked List.

**Example:**

***Input:***



***Output:***



[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Recommended Problem

Reverse a Doubly Linked List

[doubly-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=doubly-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+1 more

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[D-E-Shaw](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=D-E-Shaw&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/reverse-a-doubly-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 57.3K

Follow the given steps to solve the problem using the above approach:

* Traverse the linked list using a pointer
* Swap the prev and next pointers for all nodes
* At last, change the head pointer of the doubly linked list

Below is the implementation of the above approach:

* C
* C++
* Java
* Python3
* C#
* Javascript

# Python3 Program to reverse a doubly linked list

# A node of the doubly linked list

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

     # Constructor for empty Doubly Linked List

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function reverse a Doubly Linked List

**def** reverse(self):

        temp **=** None

        current **=** self.head

        # Swap next and prev for all nodes of

        # doubly linked list

**while** current **is not** None:

            temp **=** current.prev

            current.prev **=** current.next

            current.next **=** temp

            current **=** current.prev

        # Before changing head, check for the cases like

        # empty list and list with only one node

**if** temp **is not** None:

            self.head **=** temp.prev

    # Given a reference to the head of a list and an

    # integer,inserts a new node on the front of list

**def** push(self, new\_data):

        # 1. Allocates node

        # 2. Put the data in it

        new\_node **=** Node(new\_data)

        # 3. Make next of new node as head and

        # previous as None (already None)

        new\_node.next **=** self.head

        # 4. change prev of head node to new\_node

**if** self.head **is not** None:

            self.head.prev **=** new\_node

        # 5. move the head to point to the new node

        self.head **=** new\_node

**def** printList(self, node):

**while**(node **is not** None):

            print(node.data, end**=**' ')

            node **=** node.next

# Driver's code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    dll **=** DoublyLinkedList()

    dll.push(2)

    dll.push(4)

    dll.push(8)

    dll.push(10)

**print**("\nOriginal Linked List")

    dll.printList(dll.head)

    # Function call

    dll.reverse()

    print("\nReversed Linked List")

    dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Original Linked list  
10 8 4 2   
Reversed Linked list  
2 4 8 10

**Time Complexity:**O(N), where N denotes the number of nodes in the doubly linked list.

**Auxiliary Space:**O(1)

We can also swap data instead of pointers to reverse the Doubly Linked List. [Method used for reversing array](https://www.geeksforgeeks.org/write-a-program-to-reverse-an-array-or-string/) can be used to swap data. Swapping data can be costly compared to pointers if the size of the data item(s) is more.

**Reverse a Doubly Linked List using Stack:**

*Push the node’s data into the stack while traversing the doubly linked list, then pop out the elements from the stack and copy the value to the nodes of the linked list by again traversing it*

Follow the given steps to solve the problem using the above approach:

* Traverse the whole Linked List and  Keep pushing the node’s data into the stack
* Then keep popping the elements out of the stack and updating the Doubly Linked List

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 code for the above approach

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

**def** \_\_init\_\_(self):

        self.head **=** None

    """

    method to reverse a Doubly-Linked List using Stacks

    """

**def** reverseUsingStacks(self):

        stack **=** []

        temp **=** self.head

**while** temp **is not** None:

            stack.append(temp.data)

            temp **=** temp.next

        # Add all the elements in the stack

        # in a sequence to the stack

        temp **=** self.head

**while** temp **is not** None:

            temp.data **=** stack.pop()

            temp **=** temp.next

        # Popped all the elements and the

        # added in the linked list,

        # in a reversed order.

    """

    method to push a new item before the head

    """

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

**if** self.head **is not** None:

            self.head.prev **=** new\_node

        self.head **=** new\_node

    """

    method to traverse the doubly-linked

    list and print every node in the list

    """

**def** printList(self, node):

**while**(node **is not** None):

**print**(node.data)

            node **=** node. next

# driver's code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    dll **=** DoublyLinkedList()

    dll.push(2)

    dll.push(4)

    dll.push(8)

    dll.push(10)

**print**("original doubly-linked list")

    dll.printList(dll.head)

    # Function call

    dll.reverseUsingStacks()

**print**(" reversed doubly-linked list")

    dll.printList(dll.head)

**Output**

Original linked list   
10 8 4 2   
The reversed Linked List is   
2 4 8 10

**Time Complexity:** O(N)

**Auxiliary Space:**O(N)

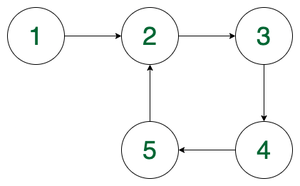
*From <*[*https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/*](https://www.geeksforgeeks.org/reverse-a-doubly-linked-list/)*>*

**Medium Questions:**

**Detect loop in a linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 14 Dec, 2022
* Read
* Discuss(118)
* Courses
* Practice
* Video

Given a linked list, check if the linked list has a loop or not. The below diagram shows a linked list with a loop.



*Linked List with Loop*

Recommended Problem

Detect Loop in linked list

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[two-pointer-algorithm](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=two-pointer-algorithm&sortBy=submissions)

+2 more

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Accolite&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+16 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/detect-loop-in-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 3.1L

**Detect loop in a linked list using**[**Hashing**](https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/)**:**

*The idea is to insert the nodes in the****hashmap****and whenever a node is encountered that is already present in the hashmap then return true.*

Follow the steps below to solve the problem:

* Traverse the list individually and keep putting the node addresses in a Hash Table.
* At any point, if NULL is reached then return false
* If the next of the current nodes points to any of the previously stored nodes in  Hash then return true.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to detect loop

# in the linked list

# Node class

**class** Node:

    # Constructor to initialize

    # the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new

    # node at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Utility function to print it

    # the linked LinkedList

**def** printList(self):

        temp **=** self.head

**while**(temp):

            print(temp.data, end**=**" ")

            temp **=** temp.next

**def** detectLoop(self):

        s **=** set()

        temp **=** self.head

**while** (temp):

            # If we already have

            # this node in hashmap it

            # means there is a cycle

            # (Because we are encountering

            # the node second time).

**if** (temp **in** s):

**return** True

            # If we are seeing the node for

            # the first time, insert it in hash

            s.add(temp)

            temp **=** temp.next

**return** False

# Driver program for testing

llist **=** LinkedList()

llist.push(20)

llist.push(4)

llist.push(15)

llist.push(10)

# Create a loop for testing

llist.head.next.next.next.next **=** llist.head

**if**(llist.detectLoop()):

**print**("Loop Found")

**else**:

**print**("No Loop ")

# This code is contributed by Gitanjali.

**Output**

Loop Found

**Time complexity:** O(N), Only one traversal of the loop is needed.

**Auxiliary Space:** O(N), N is the space required to store the value in the hashmap.

*From <*[*https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/*](https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/)*>*

**Find length of loop/cycle in given Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 25 Oct, 2022
* Read
* Discuss(32)
* Courses
* Practice
* Video

Given the head of a linked list. The task is to find if a loop exists in the linked list if **yes** then return the **length of the loop** in the linked list else return **0**.

**Examples:**

***Input:****linked list =*



***Output:****4*

*Explanation: The loop is present in the below-linked list and the length of the loop is 4.*

***Input:****linked list = 4 -> 3 -> 7 -> 9 -> 2*

***Output:****0*

Recommended Problem

Find length of Loop

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Qualcomm](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Qualcomm&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/find-length-of-loop/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 91.9K

**Approach:** Below is the idea to solve the problem:

[*Floyd’s Cycle detection algorithm*](https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/)*terminates when****fast****and****slow****pointers meet at a common point. It is also known that this common point is one of the loop nodes. Store the address of this common point in a pointer variable****ptr****. Then initialize a counter with****1****and start from the****common point****and keeps on visiting the next node and increasing the counter till the common pointer is reached again. At that point, the value of the counter will be equal to the length of the loop.*

Follow the below steps to implement the idea:

* Find the common point in the loop by using the [Floyd’s Cycle detection algorithm](https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/)
* Store the pointer in a temporary variable and keep a **count = 0**
* Traverse the linked list until the same node is reached again and increase the **count**while moving to next node.
* Print the count as length of loop

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python 3 program to find the number

# of nodes in loop in a linked list

# if loop is present

# Python Code to detect a loop and

# find the length of the loop

# Node defining class

**class** Node:

    # Function to make a node

**def** \_\_init\_\_(self, val):

        self.val **=** val

        self.next **=** None

# Linked List defining and loop

# length finding class

**class** LinkedList:

    # Function to initialize the

    # head of the linked list

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new

    # node at the end

**def** AddNode(self, val):

**if** self.head **is** None:

            self.head **=** Node(val)

**else**:

            curr **=** self.head

**while**(curr.next):

                curr **=** curr.next

            curr.next **=** Node(val)

    # Function to create a loop in the

    # Linked List. This function creates

    # a loop by connecting the last node

    # to n^th node of the linked list,

    # (counting first node as 1)

**def** CreateLoop(self, n):

        # LoopNode is the connecting node to

        # the last node of linked list

        LoopNode **=** self.head

**for** \_ **in** range(1, n):

            LoopNode **=** LoopNode.next

        # end is the last node of the Linked List

        end **=** self.head

**while**(end.next):

            end **=** end.next

        # Creating the loop

        end.next **=** LoopNode

    # Function to detect the loop and return

    # the length of the loop if the returned

    # value is zero, that means that either

    # the linked list is empty or the linked

    # list doesn't have any loop

**def** detectLoop(self):

        # if linked list is empty then there

        # is no loop, so return 0

**if** self.head **is** None:

**return** 0

        # Using Floyd’s Cycle-Finding

        # Algorithm/ Slow-Fast Pointer Method

        slow **=** self.head

        fast **=** self.head

        flag **=** 0  # to show that both slow and fast

        # are at start of the Linked List

**while**(slow **and** slow.next **and** fast **and**

              fast.next **and** fast.next.next):

**if** slow **==** fast **and** flag !**=** 0:

                # Means loop is confirmed in the

                # Linked List. Now slow and fast

                # are both at the same node which

                # is part of the loop

                count **=** 1

                slow **=** slow.next

**while**(slow !**=** fast):

                    slow **=** slow.next

                    count **+=** 1

**return** count

            slow **=** slow.next

            fast **=** fast.next.next

            flag **=** 1

**return** 0  # No loop

# Setting up the code

# Making a Linked List and adding the nodes

myLL **=** LinkedList()

myLL.AddNode(1)

myLL.AddNode(2)

myLL.AddNode(3)

myLL.AddNode(4)

myLL.AddNode(5)

# Creating a loop in the linked List

# Loop is created by connecting the

# last node of linked list to n^th node

# 1<= n <= len(LinkedList)

myLL.CreateLoop(2)

# Checking for Loop in the Linked List

# and printing the length of the loop

loopLength **=** myLL.detectLoop()

**if** myLL.head **is** None:

**print**("Linked list is empty")

**else**:

**print**(str(loopLength))

# This code is contributed by \_Ashutosh

**Output**

4

**Time complexity:**O(N), Only one traversal of the linked list is needed.

**Auxiliary Space:**O(1), As no extra space is required.

*From <*[*https://www.geeksforgeeks.org/find-length-of-loop-in-linked-list/*](https://www.geeksforgeeks.org/find-length-of-loop-in-linked-list/)*>*

**Remove duplicates from a sorted linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(182)
* Courses
* Practice
* Video

Write a function that takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

Recommended Problem

Remove duplicate element from sorted Linked List

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

[Myntra](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Myntra&sortBy=submissions)

+4 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/remove-duplicate-element-from-sorted-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.8L

**Algorithm:** Traverse the list from the head (or start) node. While traversing, compare each node with its next node. If the data of the next node is the same as the current node then delete the next node. Before we delete a node, we need to store the next pointer of the node

**Implementation:** Functions other than removeDuplicates() are just to create a linked list and test removeDuplicates().

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 program to remove duplicate

# nodes from a sorted linked list

# Node class

**class** Node:

    # Constructor to initialize

    # the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new node

    # at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Given a reference to the head of a

    # list and a key, delete the first

    # occurrence of key in linked list

**def** deleteNode(self, key):

        # Store head node

        temp **=** self.head

        # If head node itself holds the

        # key to be deleted

**if** (temp **is not** None):

**if** (temp.data **==** key):

                self.head **=** temp.next

                temp **=** None

**return**

        # Search for the key to be deleted,

        # keep track of the previous node as

        # we need to change 'prev.next'

**while**(temp **is not** None):

**if** temp.data **==** key:

**break**

            prev **=** temp

            temp **=** temp.next

        # if key was not present in

        # linked list

**if**(temp **==** None):

**return**

        # Unlink the node from linked list

        prev.next **=** temp.next

        temp **=** None

    # Utility function to print the

    # linked LinkedList

**def** printList(self):

        temp **=** self.head

**while**(temp):

            print(temp.data , end **=** ' ')

            temp **=** temp.next

    # This function removes duplicates

    # from a sorted list

**def** removeDuplicates(self):

        temp **=** self.head

**if** temp **is** None:

**return**

**while** temp.next **is not** None:

**if** temp.data **==** temp.next.data:

                new **=** temp.next.next

                temp.next **=** None

                temp.next **=** new

**else**:

                temp **=** temp.next

**return** self.head

# Driver Code

llist **=** LinkedList()

llist.push(20)

llist.push(13)

llist.push(13)

llist.push(11)

llist.push(11)

llist.push(11)

print ("Created Linked List: ")

llist.printList()

print()

print("Linked List after removing",

             "duplicate elements:")

llist.removeDuplicates()

llist.printList()

# This code is contributed by

# Dushyant Pathak.

**Output**

Linked list before duplicate removal   
 11 11 11 13 13 20  
Linked list after duplicate removal   
 11 13 20

**Time Complexity:** O(n) where n is the number of nodes in the given linked list.

**Auxiliary Space**: O(1) , as there is no extra space used.

*From <*[*https://www.geeksforgeeks.org/remove-duplicates-from-a-sorted-linked-list/*](https://www.geeksforgeeks.org/remove-duplicates-from-a-sorted-linked-list/)*>*

**Intersection of two Sorted Linked Lists**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 14 Dec, 2022
* Read
* Discuss(177)
* Courses
* Practice
* Video

Given two lists sorted in increasing order, create and return a new list representing the intersection of the two lists. The new list should be made with its own memory — the original lists should not be changed.

**Example:**

**Input:**   
First linked list: 1->2->3->4->6  
Second linked list be 2->4->6->8,   
**Output:** 2->4->6.  
The elements 2, 4, 6 are common in   
both the list so they appear in the   
intersection list.

**Input:**   
First linked list: 1->2->3->4->5  
Second linked list be 2->3->4,   
**Output:** 2->3->4  
The elements 2, 3, 4 are common in   
both the list so they appear in the   
intersection list.

Recommended Problem

Intersection of two sorted Linked lists

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Sorting](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Sorting&sortBy=submissions)

+2 more

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[D-E-Shaw](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=D-E-Shaw&sortBy=submissions)

+2 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/intersection-of-two-sorted-linked-lists/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 79K

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Method 1:** Using Dummy Node.

**Approach:**

The idea is to use a temporary dummy node at the start of the result list. The pointer tail always points to the last node in the result list, so new nodes can be added easily. The dummy node initially gives the tail a memory space to point to. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either ‘a’ or ‘b’ and adding it to the tail. When the given lists are traversed the result is in dummy. next, as the values are allocated from next node of the dummy. If both the elements are equal then remove both and insert the element to the tail. Else remove the smaller element among both the lists.

Below is the implementation of the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

''' Link list node '''

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** 0

        self.next **=** None

'''This solution uses the temporary

 dummy to build up the result list '''

**def** sortedIntersect(a, b):

    dummy **=** Node()

    tail **=** dummy;

    dummy.next **=** None;

    ''' Once one or the other

    list runs out -- we're done '''

**while** (a !**=** None **and** b !**=** None):

**if** (a.data **==** b.data):

            tail.next **=** push((tail.next), a.data);

            tail **=** tail.next;

            a **=** a.next;

            b **=** b.next;

        # advance the smaller list

**elif**(a.data < b.data):

            a **=** a.next;

**else**:

            b **=** b.next;

**return** (dummy.next);

''' UTILITY FUNCTIONS '''

''' Function to insert a node at

the beginning of the linked list '''

**def** push(head\_ref, new\_data):

    ''' allocate node '''

    new\_node **=** Node()

    ''' put in the data  '''

    new\_node.data **=** new\_data;

    ''' link the old list off the new node '''

    new\_node.next **=** (head\_ref);

    ''' move the head to point to the new node '''

    (head\_ref) **=** new\_node;

**return** head\_ref

''' Function to print nodes in

   a given linked list '''

**def** printList(node):

**while** (node !**=** None):

        print(node.data, end**=**' ')

        node **=** node.next;

''' Driver code'''

**if** \_\_name\_\_**==**'\_\_main\_\_':

    ''' Start with the empty lists '''

    a **=** None;

    b **=** None;

    intersect **=** None;

    ''' Let us create the first sorted

     linked list to test the functions

     Created linked list will be

     1.2.3.4.5.6 '''

    a **=** push(a, 6);

    a **=** push(a, 5);

    a **=** push(a, 4);

    a **=** push(a, 3);

    a **=** push(a, 2);

    a **=** push(a, 1);

    ''' Let us create the second sorted linked list

   Created linked list will be 2.4.6.8 '''

    b **=** push(b, 8);

    b **=** push(b, 6);

    b **=** push(b, 4);

    b **=** push(b, 2);

    ''' Find the intersection two linked lists '''

    intersect **=** sortedIntersect(a, b);

**print**("Linked list containing common items of a & b ");

    printList(intersect);

# This code is contributed by rutvik\_56.

**Output**

Linked list containing common items of a & b   
2 4 6

**Complexity Analysis:**

* **Time Complexity:** O(m+n) where m and n are number of nodes in first and second linked lists respectively.   
  Only one traversal of the lists are needed.
* **Auxiliary Space:** O(min(m, n)).   
  The output list can store at most min(m,n) nodes .

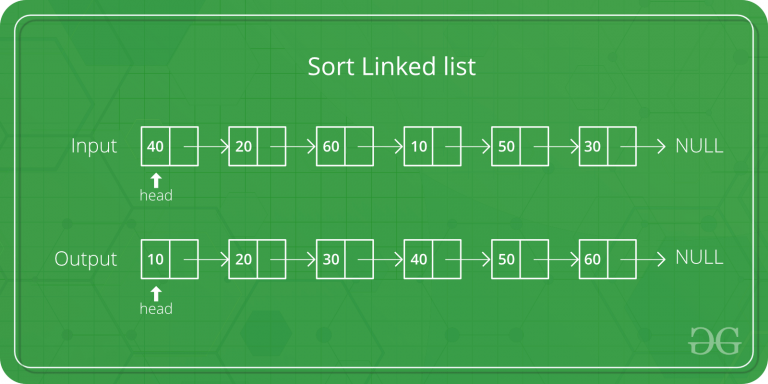
*From <*[*https://www.geeksforgeeks.org/intersection-of-two-sorted-linked-lists/*](https://www.geeksforgeeks.org/intersection-of-two-sorted-linked-lists/)*>*

**QuickSort on Singly Linked List**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 25 Oct, 2022
* Read
* Discuss(127)
* Courses
* Practice
* Video

Given a linked list, apply the [Quick sort](http://www.geeksforgeeks.org/quick-sort/) algorithm to sort the linked list. The important things about implementation are, that it changes pointers rather than swapping data.

**Example:**



Recommended Problem

Quick Sort on Linked List

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Paytm](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Paytm&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/quick-sort-on-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 14.1K

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Follow the given steps to solve the problem:

* Call partition function to get a pivot node placed at its correct position
* In the partition function, the last element is considered a pivot
* Then traverse the current list and if a node has a value greater than the pivot, then move it after the tail. If the node has a smaller value, then keep it at its current position.
* Return pivot node
* Find the tail node of the list which is on the left side of the pivot and recur for the left list
* Similarly, after the left side, recur for the list on the right side of the pivot
* Now return the head of the linked list after joining the left and the right list, as the whole linked list is now sorted

Below is the implementation of the above approach:

* C
* C++
* Java
* Python3
* C#
* Javascript

'''

sort a linked list using quick sort

'''

**class** Node:

**def** \_\_init\_\_(self, val):

        self.data **=** val

        self.next **=** None

**class** QuickSortLinkedList:

**def** \_\_init\_\_(self):

        self.head **=** None

**def** addNode(self, data):

**if** (self.head **==** None):

            self.head **=** Node(data)

**return**

        curr **=** self.head

**while** (curr.next !**=** None):

            curr **=** curr.next

        newNode **=** Node(data)

        curr.next **=** newNode

**def** printList(self, n):

**while** (n !**=** None):

**print**(n.data, end**=**" ")

            n **=** n.next

    ''' takes first and last node,but do not

    break any links in    the whole linked list'''

**def** paritionLast(self, start, end):

**if** (start **==** end **or** start **==** None **or** end **==** None):

**return** start

        pivot\_prev **=** start

        curr **=** start

        pivot **=** end.data

        '''iterate till one before the end,

        no need to iterate till the end because end is pivot'''

**while** (start !**=** end):

**if** (start.data < pivot):

                # keep tracks of last modified item

                pivot\_prev **=** curr

                temp **=** curr.data

                curr.data **=** start.data

                start.data **=** temp

                curr **=** curr.next

            start **=** start.next

        '''swap the position of curr i.e.

        next suitable index and pivot'''

        temp **=** curr.data

        curr.data **=** pivot

        end.data **=** temp

        ''' return one previous to current because

        current is now pointing to pivot '''

**return** pivot\_prev

**def** sort(self, start, end):

**if**(start **==** None **or** start **==** end **or** start **==** end.next):

**return**

        # split list and partition recurse

        pivot\_prev **=** self.paritionLast(start, end)

        self.sort(start, pivot\_prev)

        '''

        if pivot is picked and moved to the start,

        that means start and pivot is same

        so pick from next of pivot

        '''

**if**(pivot\_prev !**=** None **and** pivot\_prev **==** start):

            self.sort(pivot\_prev.next, end)

        # if pivot is in between of the list,start from next of pivot,

        # since we have pivot\_prev, so we move two nodes

**elif** (pivot\_prev !**=** None **and** pivot\_prev.next !**=** None):

            self.sort(pivot\_prev.next.next, end)

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    ll **=** QuickSortLinkedList()

    ll.addNode(30)

    ll.addNode(3)

    ll.addNode(4)

    ll.addNode(20)

    ll.addNode(5)

    N **=** ll.head

**while** (N.next !**=** None):

        N **=** N.next

**print**("\nLinked List before sorting")

    ll.printList(ll.head)

    # Function call

    ll.sort(ll.head, N)

    print("\nLinked List after sorting")

    ll.printList(ll.head)

    # This code is contributed by humpheykibet.

**Output**

Linked List before sorting   
30 3 4 20 5   
Linked List after sorting   
3 4 5 20 30

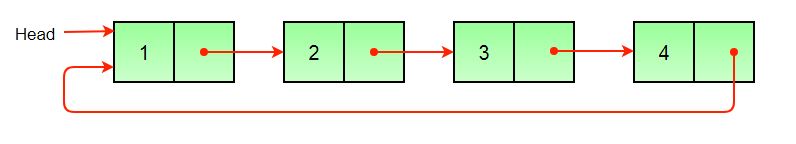
**Time Complexity:**O(N \* log N), It takes O(N2) time in the worst case and O(N log N) in the average or best case.

**Auxiliary Space:** O(N), As extra space is used in the recursion call stack.

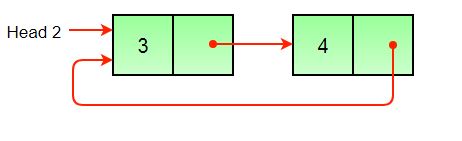
*From <*[*https://www.geeksforgeeks.org/quicksort-on-singly-linked-list/*](https://www.geeksforgeeks.org/quicksort-on-singly-linked-list/)*>*

**Split a Circular Linked List into two halves**

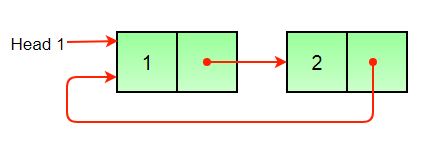
* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 24 Jun, 2022
* Read
* Discuss(80)
* Courses
* Practice
* Video



Original Linked List



Result Linked List 1



Result Linked List 2

If there are **odd number of nodes**, then first list should contain one extra.

Recommended Problem

Split a Circular Linked List into two halves

[circular-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=circular-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+1 more

[Yahoo](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Yahoo&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/split-a-circular-linked-list-into-two-halves/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 42K

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Thanks to [Geek4u](https://www.geeksforgeeks.org/forum/topic/splitting-a-circular-doubly-linked-list#post-335) for suggesting the algorithm.

1) Store the mid and last pointers of the circular linked list using tortoise and hare algorithm.

2) Make the second half circular.

3) Make the first half circular.

4) Set head (or start) pointers of the two linked lists.

In the below implementation, if there are odd nodes in the given circular linked list then the first result list has 1 more node than the second result list.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to split circular linked list into two halves

# A node structure

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# Class to create a new  Circular Linked list

**class** CircularLinkedList:

    # Constructor to create a empty circular linked list

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a node at the beginning of a

    # circular linked list

**def** push(self, data):

        ptr1 **=** Node(data)

        temp **=** self.head

        ptr1.next **=** self.head

        # If linked list is not None then set the next of

        # last node

**if** self.head **is not** None:

**while**(temp.next !**=** self.head):

                temp **=** temp.next

            temp.next **=** ptr1

**else**:

            ptr1.next **=** ptr1 # For the first node

        self.head **=** ptr1

    # Function to print nodes in a given circular linked list

**def** printList(self):

        temp **=** self.head

**if** self.head **is not** None:

**while**(True):

                print ("%d" **%**(temp.data),end**=**' ')

                temp **=** temp.next

**if** (temp **==** self.head):

**break**

    # Function to split a list (starting with head) into

    # two lists. head1 and head2 are the head nodes of the

    # two resultant linked lists

**def** splitList(self, head1, head2):

        slow\_ptr **=** self.head

        fast\_ptr **=** self.head

**if** self.head **is** None:

**return**

        # If there are odd nodes in the circular list then

        # fast\_ptr->next becomes head and for even nodes

        # fast\_ptr->next->next becomes head

**while**(fast\_ptr.next !**=** self.head **and**

            fast\_ptr.next.next !**=** self.head ):

            fast\_ptr **=** fast\_ptr.next.next

            slow\_ptr **=** slow\_ptr.next

        # If there are even elements in list then

        # move fast\_ptr

**if** fast\_ptr.next.next **==** self.head:

            fast\_ptr **=** fast\_ptr.next

        # Set the head pointer of first half

        head1.head **=** self.head

        # Set the head pointer of second half

**if** self.head.next !**=** self.head:

            head2.head **=** slow\_ptr.next

        # Make second half circular

        fast\_ptr.next **=** slow\_ptr.next

        # Make first half circular

        slow\_ptr.next **=** self.head

# Driver program to test above functions

# Initialize lists as empty

head **=** CircularLinkedList()

head1 **=** CircularLinkedList()

head2 **=** CircularLinkedList()

head.push(12)

head.push(56)

head.push(2)

head.push(11)

print ("Original Circular Linked List")

head.printList()

# Split the list

head.splitList(head1 , head2)

**print** ("\nFirst Circular Linked List")

head1.printList()

**print** ("\nSecond Circular Linked List")

head2.printList()

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output:**

Original Circular Linked List  
11 2 56 12   
First Circular Linked List  
11 2   
Second Circular Linked List  
56 12

**Time Complexity:** O(n) As we are only moving once through the list

**Auxiliary Space:** O(1) As no extra space is used

Please write comments if you find any bug in above code/algorithm, or find other ways to solve the same problem

*From <*[*https://www.geeksforgeeks.org/split-a-circular-linked-list-into-two-halves/*](https://www.geeksforgeeks.org/split-a-circular-linked-list-into-two-halves/)*>*

**Deletion from a Circular Linked List**

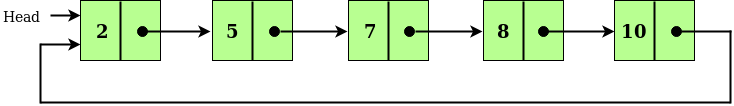
* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(31)
* Courses
* Practice
* Video

We have already discussed circular linked list and traversal in a circular linked list in the below articles:

[Introduction to circular linked list](https://www.geeksforgeeks.org/circular-linked-list/)

[Traversal in a circular linked list](https://www.geeksforgeeks.org/circular-linked-list-set-2-traversal/)

In this article, we will learn about deleting a node from a circular linked list. Consider the linked list as shown below:



We will be given a node and our task is to delete that node from the circular linked list.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Examples:**

Input : 2->5->7->8->10->(head node)  
 data = 5  
Output : 2->7->8->10->(head node)

Input : 2->5->7->8->10->(head node)  
 7  
Output : 2->5->8->10->(head node)

**Algorithm**

**Case 1**: List is empty.

* If the list is empty we will simply return.

**Case 2**:List is not empty

* If the list is not empty then we define two pointers **curr**and **prev** and initialize the pointer **curr** with the **head**node.
* Traverse the list using **curr** to find the node to be deleted and before moving to curr to the next node, every time set prev = curr.
* If the node is found, check if it is the only node in the list. If yes, set head = NULL and free(curr).
* If the list has more than one node, check if it is the first node of the list. Condition to check this( curr == head). If yes, then move prev until it reaches the last node. After prev reaches the last node, set head = head -> next and prev -> next = head. Delete curr.
* If curr is not the first node, we check if it is the last node in the list. Condition to check this is (curr -> next == head).
* If curr is the last node. Set prev -> next = head and delete the node curr by free(curr).
* If the node to be deleted is neither the first node nor the last node, then set prev -> next = curr -> next and delete curr.

Recommended Problem

Deletion and Reverse in Linked List

[circular linked list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=circular%20linked%20list&sortBy=submissions)

[circular-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=circular-linked-list&sortBy=submissions)

+2 more

[Intuit](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Intuit%20&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/deletion-and-reverse-in-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 16.9K

**Complete program to demonstrate deletion in Circular Linked List:**

* C++14
* C
* Java
* Python
* C#
* Javascript

# Python program to delete a given key from

# linked list.

# Node of a doubly linked list

**class** Node:

**def** \_\_init\_\_(self, next **=** None, data **=** None):

        self.next **=** next

        self.data **=** data

# Function to insert a node at the beginning of

# a Circular linked list

**def** push(head\_ref, data):

    # Create a new node and make head as next

    # of it.

    ptr1 **=** Node()

    ptr1.data **=** data

    ptr1.next **=** head\_ref

    # If linked list is not None then set the

    # next of last node

**if** (head\_ref !**=** None) :

        # Find the node before head and update

        # next of it.

        temp **=** head\_ref

**while** (temp.next !**=** head\_ref):

            temp **=** temp.next

        temp.next **=** ptr1

**else**:

        ptr1.next **=** ptr1 # For the first node

    head\_ref **=** ptr1

**return** head\_ref

# Function to print nodes in a given

# circular linked list

**def** printList( head):

    temp **=** head

**if** (head !**=** None) :

**while**(True) :

**print**( temp.data, end **=** " ")

            temp **=** temp.next

**if** (temp **==** head):

**break**

    print()

# Function to delete a given node from the list

**def** deleteNode( head, key) :

    # If linked list is empty

**if** (head **==** None):

**return**

    # If the list contains only a single node

**if**((head).data **==** key **and** (head).next **==** head):

        head **=** None

    last **=** head

    d **=** None

    # If head is to be deleted

**if**((head).data **==** key) :

        # Find the last node of the list

**while**(last.next !**=** head):

            last **=** last.next

        # Point last node to the next of head i.e.

        # the second node of the list

        last.next **=** (head).next

        head **=** last.next

    # Either the node to be deleted is not found

    # or the end of list is not reached

**while**(last.next !**=** head **and** last.next.data !**=** key) :

        last **=** last.next

    # If node to be deleted was found

**if**(last.next.data **==** key) :

        d **=** last.next

        last.next **=** d.next

**else**:

        print("no such keyfound")

**return** head

# Driver code

# Initialize lists as empty

head **=** None

# Created linked list will be 2.5.7.8.10

head **=** push(head, 2)

head **=** push(head, 5)

head **=** push(head, 7)

head **=** push(head, 8)

head **=** push(head, 10)

print("List Before Deletion: ")

printList(head)

head **=** deleteNode(head, 7)

**print**( "List After Deletion: ")

printList(head)

# This code is contributed by Arnab Kundu

**Output**

List Before Deletion: 10 8 7 5 2   
List After Deletion: 10 8 5 2

**Time Complexity:** O(n), Worst case occurs when the element to be deleted is the last element and we need to move through the whole list.

**Auxiliary Space:** O(1), As constant extra space is used.

*From <*[*https://www.geeksforgeeks.org/deletion-circular-linked-list/*](https://www.geeksforgeeks.org/deletion-circular-linked-list/)*>*

**Convert singly linked list into circular linked list**

* Difficulty Level : [Basic](https://www.geeksforgeeks.org/basic/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given a singly linked list, we have to convert it into circular linked list. For example, we have been given a singly linked list with four nodes and we want to convert this singly linked list into circular linked list.



The above singly linked list is converted into circular linked list.



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach:** The idea is to traverse the singly linked list and check if the node is the last node or not. If the node is the last node i.e pointing to NULL then make it point to the starting node i.e head node. Below is the implementation of this approach.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program for converting

# singly linked list into

# circular linked list.

**import** sys

# Linked list node

**class** Node:

**def** \_\_init\_\_(self,data):

        self.data **=** data

        self.next **=** None

**def** push(head, data):

**if not** head:

**return** Node(data)

    # Allocate dynamic memory

    # for newNode.

    # Assign the data into newNode.

    newNode **=** Node(data)

    # newNode.next assign the

    # address of head node.

    newNode.next **=** head

    # newNode become the headNode.

    head **=** newNode

**return** head

# Function that convert

# singly linked list into

# circular linked list.

**def** circular(head):

    # declare a node variable

    # start and assign head

    # node into start node.

    start **=** head

    # check that while head.next

    # not equal to null then head

    # points to next node.

**while**(head.next **is not** None):

        head **=** head.next

    # if head.next points to null

    # then start assign to the

    # head.next node.

    head.next **=** start

**return** start

# Function that display the elements

# of circular linked list.

**def** displayList(node):

    start **=** node

**while**(node.next **is not** start):

**print**("{} ".format(node.data),end**=**"")

        node**=**node.next

    # Display the last node of

    # circular linked list.

    print("{} ".format(node.data),end**=**"")

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Start with empty list

    head**=**None

    # Using push() function to

    # convert singly linked list

    # 17.22.13.14.15

    head**=**push(head,15)

    head**=**push(head,14)

    head**=**push(head,13)

    head**=**push(head,22)

    head**=**push(head,17)

    # Call the circular\_list function

    # that convert singly linked

    # list to circular linked list.

    circular(head)

    print("Display List:")

    displayList(head)

# This Code is Contributed By Vikash Kumar 37

**Output**

Display list:   
17 22 13 14 15

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Time Complexity:** **O(n),**As we need to move through the whole list to get hold of the last node.

**Auxiliary Space: O(1),**As constant extra space is used.

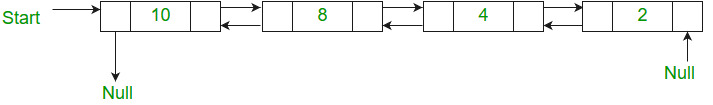
*From <*[*https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/*](https://www.geeksforgeeks.org/convert-singly-linked-list-circular-linked-list/)*>*

**Merge Sort for Doubly Linked List**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(27)
* Courses
* Practice
* Video

Given a doubly linked list, write a function to sort the doubly linked list in increasing order using merge sort.

For example the following doubly linked list should be changed to {2, 4, 8, 10}.



Recommended Problem

Merge Sort on Doubly Linked List

[doubly-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=doubly-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+4 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/merge-sort-on-doubly-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 14.5K

[Merge sort for singly linked list](https://www.geeksforgeeks.org/merge-sort-for-linked-list/) is already discussed. The important change here is to modify the previous pointers also when merging two lists.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Below is the implementation of merge sort for doubly linked list.

* C++
* C
* Java
* Python3
* C#
* Javascript

# Program for merge sort on doubly linked list

# A node of the doubly linked list

**class** Node:

    # Constructor to create a new node

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

     # Constructor for empty Doubly Linked List

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to merge two linked list

**def** merge(self, first, second):

        # If first linked list is empty

**if** first **is** None:

**return** second

        # If second linked list is empty

**if** second **is** None:

**return** first

        # Pick the smaller value

**if** first.data < second.data:

            first.next **=** self.merge(first.next, second)

            first.next.prev **=** first

            first.prev **=** None

**return** first

**else**:

            second.next **=** self.merge(first, second.next)

            second.next.prev **=** second

            second.prev **=** None

**return** second

    # Function to do merge sort

**def** mergeSort(self, tempHead):

**if** tempHead **is** None:

**return** tempHead

**if** tempHead.next **is** None:

**return** tempHead

        second **=** self.split(tempHead)

        # Recur for left and right halves

        tempHead **=** self.mergeSort(tempHead)

        second **=** self.mergeSort(second)

        # Merge the two sorted halves

**return** self.merge(tempHead, second)

    # Split the doubly linked list (DLL) into two DLLs

    # of half sizes

**def** split(self, tempHead):

        fast **=** slow **=**  tempHead

**while**(True):

**if** fast.next **is** None:

**break**

**if** fast.next.next **is** None:

**break**

            fast **=** fast.next.next

            slow **=** slow.next

        temp **=** slow.next

        slow.next **=** None

**return** temp

    # Given a reference to the head of a list and an

    # integer,inserts a new node on the front of list

**def** push(self, new\_data):

        # 1. Allocates node

        # 2. Put the data in it

        new\_node **=** Node(new\_data)

        # 3. Make next of new node as head and

        # previous as None (already None)

        new\_node.next **=** self.head

        # 4. change prev of head node to new\_node

**if** self.head **is not** None:

            self.head.prev **=** new\_node

        # 5. move the head to point to the new node

        self.head **=** new\_node

**def** printList(self, node):

        temp **=** node

        print ("Forward Traversal using next pointer")

**while**(node **is not** None):

            print (node.data,end**=**" ")

            temp **=** node

            node **=** node.next

**print** ("\nBackward Traversal using prev pointer")

**while**(temp):

**print** (temp.data,end**=**" ")

            temp **=** temp.prev

# Driver program to test the above functions

dll **=** DoublyLinkedList()

dll.push(5)

dll.push(20);

dll.push(4);

dll.push(3);

dll.push(30)

dll.push(10);

dll.head **=** dll.mergeSort(dll.head)

**print** ("Linked List after sorting")

dll.printList(dll.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck\_007)

**Output**

Linked List after sorting  
Forward Traversal using next pointer  
3 4 5 10 20 30   
Backward Traversal using prev pointer  
30 20 10 5 4 3

Thanks to Goku for providing above implementation in a comment [here](https://www.geeksforgeeks.org/quicksort-for-linked-list/).

**Time Complexity:**Time complexity of the above implementation is same as time complexity of [MergeSort for arrays](https://www.geeksforgeeks.org/merge-sort/). It takes Θ(nLogn) time.

**Auxiliary Space:**O(1). We are only using constant amount of extra space.

*From <*[*https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/*](https://www.geeksforgeeks.org/merge-sort-for-doubly-linked-list/)*>*

**Find pairs with given sum in doubly linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss
* Courses
* Practice
* Video

Given a sorted doubly linked list of positive distinct elements, the task is to find pairs in a doubly-linked list whose sum is equal to given value x, without using any extra space?

**Example:**

Input : head : 1 <-> 2 <-> 4 <-> 5 <-> 6 <-> 8 <-> 9  
 x = 7  
Output: (6, 1), (5,2)

The expected time complexity is O(n) and auxiliary space is O(1).

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

A **simple approach** for this problem is to one by one pick each node and find a second element whose sum is equal to x in the remaining list by traversing in the forward direction. The time complexity for this problem will be O(n^2), n is the total number of nodes in the doubly linked list.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

*An****efficient solution****for this problem is the same as*[*this*](https://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whether-or-not-there-exist-two-elements-in-s-whose-sum-is-exactly-x/)*article. Here is the algorithm :*

* *Initialize two pointer variables to find the candidate elements in the sorted doubly linked list. Initialize****first****with the start of the doubly linked list i.e;****first=head****and initialize****second****with the last node of the doubly linked list i.e;****second=last\_node****.*
* *We initialize****first****and****second****pointers as first and last nodes. Here we don’t have random access, so to find the second pointer, we traverse the list to initialize the second.*
* *If current sum of****first****and****second****is less than x, then we move****first****in forward direction. If current sum of****first****and****second****element is greater than x, then we move****second****in backward direction.*
* *Loop termination conditions are also different from arrays. The loop terminates when two pointers cross each other (second->next = first), or they become the same (first == second).*
* *The case when no pairs are present will be handled by the condition “first==second”*

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find a pair with

# given sum x.

# Structure of node of doubly linked list

**class** Node:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.next **=** None

        self.prev **=** None

# Function to find pair whose sum

# equal to given value x.

**def** pairSum(head, x):

    # Set two pointers, first to the

    # beginning of DLL and second to

    # the end of DLL.

    first **=** head

    second **=** head

**while** (second.next !**=** None):

        second **=** second.next

    # To track if we find a pair or not

    found **=** False

    # The loop terminates when they

    # cross each other (second.next ==

    # first), or they become same

    # (first == second)

**while** (first !**=** second **and** second.next !**=** first):

        # Pair found

**if** ((first.data **+** second.data) **==** x):

            found **=** True

**print**("(", first.data, ",",

                       second.data, ")")

            # Move first in forward direction

            first **=** first.next

            # Move second in backward direction

            second **=** second.prev

**else**:

**if** ((first.data **+** second.data) < x):

                first **=** first.next

**else**:

                second **=** second.prev

    # If pair is not present

**if** (found **==** False):

**print**("No pair found")

# A utility function to insert a new node

# at the beginning of doubly linked list

**def** insert(head, data):

    temp **=** Node(data)

**if not** head:

        head **=** temp

**else**:

        temp.next **=** head

        head.prev **=** temp

        head **=** temp

**return** head

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    head **=** None

    head **=** insert(head, 9)

    head **=** insert(head, 8)

    head **=** insert(head, 6)

    head **=** insert(head, 5)

    head **=** insert(head, 4)

    head **=** insert(head, 2)

    head **=** insert(head, 1)

    x **=** 7

    pairSum(head, x)

# This code is contributed by mohit kumar 29

**Output**

(1, 6)  
(2, 5)

**Time complexity : O(n)**

**Auxiliary space : O(1)**

If linked list is not sorted, then we can sort the list as a first step. But in that case overall time complexity would become O(n Log n). We can use Hashing in such cases if extra space is not a constraint. The hashing based solution is same as method 2 [here](https://www.geeksforgeeks.org/write-a-c-program-that-given-a-set-a-of-n-numbers-and-another-number-x-determines-whether-or-not-there-exist-two-elements-in-s-whose-sum-is-exactly-x/).

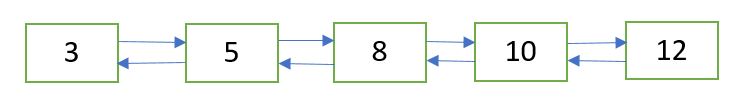
*From <*[*https://www.geeksforgeeks.org/find-pairs-given-sum-doubly-linked-list/*](https://www.geeksforgeeks.org/find-pairs-given-sum-doubly-linked-list/)*>*

**Insert value in sorted way in a sorted doubly linked list**

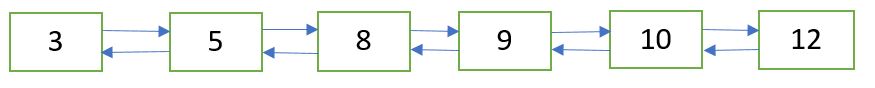
* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 14 Jul, 2022
* Read
* Discuss(4)
* Courses
* Practice
* Video

Given a sorted doubly linked list and a value to insert, write a function to insert the value in sorted way.

Initial doubly linked list



Doubly Linked List after insertion of 9



Recommended Problem

Insert in Sorted way in a Sorted DLL

[doubly-linked-list](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=doubly-linked-list&sortBy=submissions)

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

+1 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/insert-in-sorted-way-in-a-sorted-dll/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 18.5K

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Algorithm:**

Let input doubly linked list is sorted in increasing order.  New node passed to the function contains data in the data part and previous and next link are set to NULL.

**sortedInsert(head\_ref, newNode)**  
 if (head\_ref == NULL)  
 head\_ref = newNode  
   
 else if head\_ref->data >= newNode->data  
 newNode->next = head\_ref  
 newNode->next->prev = newNode  
 head\_ref = newNode   
   
 else  
 Initialize current = head\_ref  
 while (current->next != NULL and  
 current->next->data < newNode->data)  
 current = current->next  
   
 newNode->next = current->next  
 if current->next != NULL  
 newNode->next->prev = newNode  
   
 current->next = newNode  
 newNode->prev = current

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 implementation to insert

# value in sorted way in a sorted

# doubly linked list

# Node of a doubly linked list

**class** Node:

    # Constructor to initialize

    # the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

        self.prev **=** None

**class** DoublyLinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to create and return a

    # new node of a doubly linked list

**def** getNode(self, data):

**return** Node(data)

    # Function to insert a new node in

    # sorted way in a sorted doubly linked list

**def** sortedInsert(self, data):

        new\_node **=** self.getNode(data)

        # If the list is empty

**if** self.head **is** None:

            self.head **=** new\_node

        # If the node is to be inserted at

        # the beginning of the doubly linked list

**elif** self.head.data >**=** new\_node.data:

            new\_node.next **=** self.head

            new\_node.next.prev **=** new\_node

            self.head **=** new\_node

**else**:

            current **=** self.head

            # Locate the node after which

            # the new node  is to be inserted

**while** ((current.next **is not** None) **and**

                   (current.next.data < new\_node.data)):

                current **=** current.next

            # Make the appropriate links

            new\_node.next **=** current.next

            # If the new node is not inserted

            # at the end of the list

**if** current.next **is not** None:

                new\_node.next.prev **=** new\_node

            current.next **=** new\_node

            new\_node.prev **=** current

    # Function to print the doubly linked list

**def** printList(self):

        node **=** self.head

**while** node:

**print**(str(node.data), end **=** " ")

            node **=** node.next

# Driver code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Insert the following nodes

    # in sorted way

    llist **=** DoublyLinkedList()

    llist.sortedInsert(8)

    llist.sortedInsert(5)

    llist.sortedInsert(3)

    llist.sortedInsert(10)

    llist.sortedInsert(12)

    llist.sortedInsert(9)

**print**("Created Doubly Linked List")

    llist.printList()

# This code is contributed by Siddhartha Pramanik

**Output**

Created Doubly Linked Listn3 5 8 9 10 12

**Time Complexity:** **O(n)**

**Auxiliary Space: O(n)**

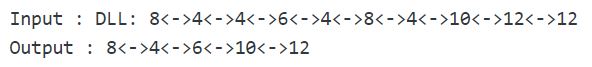
*From <*[*https://www.geeksforgeeks.org/insert-value-sorted-way-sorted-doubly-linked-list/*](https://www.geeksforgeeks.org/insert-value-sorted-way-sorted-doubly-linked-list/)*>*

**Remove duplicates from an unsorted doubly linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(3)
* Courses
* Practice
* Video

Given an unsorted doubly linked list containing **n** nodes. The problem is to remove duplicate nodes from the given list.

**Examples:**



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Method 1 (Naive Approach):**

This is the simplest way where two loops are used. The outer loop is used to pick the elements one by one and the inner loop compares the picked element with the rest of the elements.

**Implementation:**

* C++
* Java
* Python
* C#
* Javascript

# Python implementation to remove duplicates

# from an unsorted doubly linked list

# Node of a linked list

**class** Node:

**def** \_\_init\_\_(self, data **=** None, next **=** None):

        self.next **=** next

        self.data **=** data

# Function to delete a node in a Doubly Linked List.

# head\_ref -. pointer to head node pointer.

# del -. pointer to node to be deleted.

**def** deleteNode(head\_ref,del\_):

    # base case

**if** (head\_ref **==** None **or** del\_ **==** None):

**return** head\_ref

    # If node to be deleted is head node

**if** (head\_ref **==** del\_):

        head\_ref **=** del\_.next

    # Change next only if node to be deleted

    # is NOT the last node

**if** (del\_.next !**=** None):

        del\_.next.prev **=** del\_.prev

    # Change prev only if node to be deleted

    # is NOT the first node

**if** (del\_.prev !**=** None):

        del\_.prev.next **=** del\_.next

**return** head\_ref

# function to remove duplicates from

# an unsorted doubly linked list

**def** removeDuplicates( head\_ref):

    # if DLL is empty or if it contains only

    # a single node

**if** ((head\_ref) **==** None **or** (head\_ref).next **==** None):

**return** head\_ref

    ptr1 **=** head\_ref

    ptr2 **=** None

    # pick elements one by one

**while**(ptr1 !**=** None) :

        ptr2 **=** ptr1.next

        # Compare the picked element with the

        # rest of the elements

**while** (ptr2 !**=** None):

            # if duplicate, then delete it

**if** (ptr1.data **==** ptr2.data):

                # store pointer to the node next to 'ptr2'

                next **=** ptr2.next

                # delete node pointed to by 'ptr2'

                head\_ref **=** deleteNode(head\_ref, ptr2)

                # update 'ptr2'

                ptr2 **=** next

            # else simply move to the next node

**else**:

                ptr2 **=** ptr2.next

        ptr1 **=** ptr1.next

**return** head\_ref

# Function to insert a node at the beginning

# of the Doubly Linked List

**def** push( head\_ref, new\_data):

    # allocate node

    new\_node **=** Node()

    # put in the data

    new\_node.data **=** new\_data

    # since we are adding at the beginning,

    # prev is always None

    new\_node.prev **=** None

    # link the old list off the new node

    new\_node.next **=** (head\_ref)

    # change prev of head node to new node

**if** ((head\_ref) !**=** None):

        (head\_ref).prev **=** new\_node

    # move the head to point to the new node

    (head\_ref) **=** new\_node

**return** head\_ref

# Function to print nodes in a

# given doubly linked list

**def** printList( head):

    # if list is empty

**if** (head **==** None):

**print**("Doubly Linked list empty")

**while** (head !**=** None):

**print**( head.data ,end**=** " ")

        head **=** head.next

# Driver Code

head **=** None

# Create the doubly linked list:

# 8<.4<.4<.6<.4<.8<.4<.10<.12<.12

head **=** push(head, 12)

head **=** push(head, 12)

head **=** push(head, 10)

head **=** push(head, 4)

head **=** push(head, 8)

head **=** push(head, 4)

head **=** push(head, 6)

head **=** push(head, 4)

head **=** push(head, 4)

head **=** push(head, 8)

print("Original Doubly linked list:")

printList(head)

# remove duplicate nodes \*/

head**=**removeDuplicates(head)

print("\nDoubly linked list after removing duplicates:")

printList(head)

# This code is contributed by Arnab Kundu

**Output**

Original Doubly linked list:n8 4 4 6 4 8 4 10 12 12   
Doubly linked list after removing duplicates:n8 4 6 10 12

**Time Complexity:** **O(n2)**

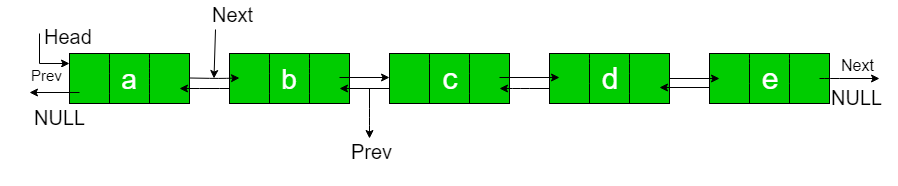
**Auxiliary Space: O(1)**

*From <*[*https://www.geeksforgeeks.org/remove-duplicates-unsorted-doubly-linked-list/*](https://www.geeksforgeeks.org/remove-duplicates-unsorted-doubly-linked-list/)*>*

**Rotate Doubly linked list by N nodes**

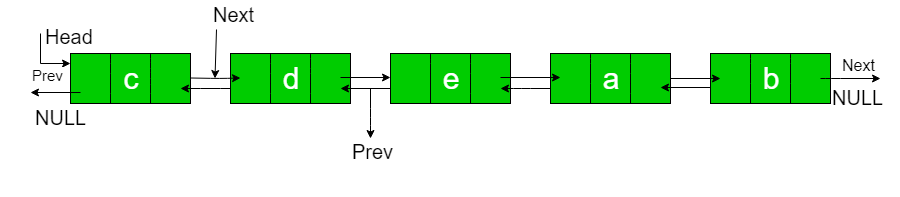
* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(18)
* Courses
* Practice
* Video

Given a doubly-linked list, rotate the linked list counter-clockwise by N nodes. Here N is a given positive integer and is smaller than the count of nodes in linked list.



N = 2

Rotated List:



**Examples:**

**Input : a b c d e**  N = 2  
**Output : c d e a b**

**Input : a b c d e f g h**  N = 4  
**Output** : **e f g h a b c d**

**Asked in**[Amazon](https://www.geeksforgeeks.org/amazon-interview-experience-set-424-sde-2/)

Recommended Practice

[Please try your approach on IDE first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

[Try It!](https://ide.geeksforgeeks.org/)

**Solution 1:**

* C++
* Java
* Python3
* C#
* Javascript

# Node of a doubly linked list

**class** Node:

**def** \_\_init\_\_(self, next **=** None,

                       prev **=** None, data **=** None):

        self.next **=** next # reference to next node in DLL

        self.prev **=** prev # reference to previous node in DLL

        self.data **=** data

**def** push(head, new\_data):

    new\_node **=** Node(data **=** new\_data)

    new\_node.next **=** head

    new\_node.prev **=** None

**if** head **is not** None:

        head.prev **=** new\_node

    head **=** new\_node

**return** head

**def** printList(head):

    node **=** head

**print**("Given linked list")

**while**(node **is not** None):

**print**(node.data, end **=** " "),

        last **=** node

        node **=** node.next

**def** rotate(start, N):

**if** N **==** 0 :

**return**

    # Let us understand the below code

    # for example N = 2 and

    # list = a <-> b <-> c <-> d <-> e.

    current **=** start

    # current will either point to Nth

    # or None after this loop. Current

    # will point to node 'b' in the

    # above example

    count **=** 1

**while** count < N **and** current !**=** None :

        current **=** current.next

        count **+=** 1

    # If current is None, N is greater

    # than or equal to count of nodes

    # in linked list. Don't change the

    # list in this case

**if** current **==** None :

**return**

    # current points to Nth node. Store

    # it in a variable. NthNode points to

    # node 'b' in the above example

    NthNode **=** current

    # current will point to last node

    # after this loop current will point

    # to node 'e' in the above example

**while** current.next !**=** None :

        current **=** current.next

    # Change next of last node to previous

    # head. Next of 'e' is now changed to

    # node 'a'

    current.next **=** start

    # Change prev of Head node to current

    # Prev of 'a' is now changed to node 'e'

    start.prev **=** current

    # Change head to (N+1)th node

    # head is now changed to node 'c'

    start **=** NthNode.next

    # Change prev of New Head node to None

    # Because Prev of Head Node in Doubly

    # linked list is None

    start.prev **=** None

    # change next of Nth node to None

    # next of 'b' is now None

    NthNode.next **=** None

**return** start

# Driver Code

**if** \_\_name\_\_ **==** "\_\_main\_\_":

    head **=** None

    head **=** push(head, 'e')

    head **=** push(head, 'd')

    head **=** push(head, 'c')

    head **=** push(head, 'b')

    head **=** push(head, 'a')

    printList(head)

**print**("\n")

    N **=** 2

    head **=** rotate(head, N)

    printList(head)

# This code is contributed by vinayak sharma

**Output**

Before Rotation :   
a-->b-->c-->d-->e-->NULL

After Rotation :   
c-->d-->e-->a-->b-->NULL

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Time Complexity: O(N)**

**Space Complexity: O(1)**

*From <*[*https://www.geeksforgeeks.org/rotate-doubly-linked-list-n-nodes/*](https://www.geeksforgeeks.org/rotate-doubly-linked-list-n-nodes/)*>*

**Given only a pointer/reference to a node to be deleted in a singly linked list, how do you delete it?**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 03 Jul, 2022
* Read
* Discuss(292)
* Courses
* Practice
* Video

Given a pointer to a node to be deleted, delete the node. Note that we don’t have a pointer to the head node.

Recommended Problem

Delete nodes having greater value on right

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon%20&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/delete-nodes-having-greater-value-on-right/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 73.6K

A **simple solution**is to traverse the linked list until you find the node you want to delete. But this solution requires a pointer to the head node, which contradicts the problem statement.

The **fast solution** is to copy the data from the next node to the node to be deleted and delete the next node. Something like the following.

// Find next node using next pointer  
struct Node \*temp = node\_ptr->next;

// Copy data of next node to this node  
node\_ptr->data = temp->data;

// Unlink next node  
node\_ptr->next = temp->next;

// Delete next node  
free(temp);

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Program:**

* C++
* C
* Java
* Python
* C#
* Javascript

# a class to define a node with

# data and next pointer

**class** Node():

    # constructor to initialize a new node

**def** \_\_init\_\_(self, val **=** None):

        self.data **=** val

        self.next **=** None

# push a node to the front of the list

**def** push(head, val):

    # allocate new node

    newnode **=** Node(val)

    # link the first node of the old list to the new node

    newnode.next **=** head.next

    # make the new node as head of the linked list

    head.next **=** newnode

# function to print the list

**def** print\_list(head):

    temp **=** head.next

**while**(temp !**=** None):

**print**(temp.data, end **=** ' ')

        temp **=** temp.next

**print**()

# function to delete the node

# the main logic is in this

**def** delete\_node(node):

    prev **=** Node()

**if**(node **==** None):

**return**

**else**:

**while**(node.next !**=** None):

            node.data **=** node.next.data

            prev **=** node

            node **=** node.next

        prev.next **=** None

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # allocate an empty header node

    # this is a node that simply points to the

    # first node in the list

    head **=** Node()

    # construct the below linked list

    # 1->12->1->4->1

    push(head, 1)

    push(head, 4)

    push(head, 1)

    push(head, 12)

    push(head, 1)

**print**('list before deleting:')

    print\_list(head)

    # deleting the first node in the list

    delete\_node(head.next)

**print**('list after deleting: ')

    print\_list(head)

# This code is contributed by Adith Bharadwaj

**Output:**

Before deleting   
1 12 1 4 1   
After deleting   
12 1 4 1

**Time Complexity:**

* **For printing linked list:**O(N)
* **For inserting node:**O(1)
* **For deleting node:**O(N)
* **Auxiliary Space:**O(1)

**This solution doesn’t work if the node to be deleted is the last node of the list.** To make this solution work, we can mark the end node as a dummy node. But the programs/functions that are using this function should also be modified.

Exercise: Try this problem with the doubly linked list.

**One line in the function deletenode():**

* C++
* Java
* Python3
* C#
* Javascript

**def** deleteNode(Node Node):

   Node **=** (Node.next);

# This code is contributed

*From <*[*https://www.geeksforgeeks.org/given-only-a-pointer-to-a-node-to-be-deleted-in-a-singly-linked-list-how-do-you-delete-it/*](https://www.geeksforgeeks.org/given-only-a-pointer-to-a-node-to-be-deleted-in-a-singly-linked-list-how-do-you-delete-it/)*>*

**Modify contents of Linked List**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 28 Nov, 2022
* Read
* Discuss(21)
* Courses
* Practice
* Video

Given a singly linked list containing **n** nodes. Modify the value of first half nodes such that 1st node’s new value is equal to the last node’s value minus first node’s current value, 2nd node’s new value is equal to the second last node’s value minus 2nd node’s current value, likewise for first half nodes. If **n** is odd then the value of the middle node remains unchanged.

(No extra memory to be used).

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Examples:**

Input : 10 -> 4 -> 5 -> 3 -> 6  
Output : 4 -> 1 -> 5 -> 3 -> 6

Input : 2 -> 9 -> 8 -> 12 -> 7 -> 10  
Output : -8 -> 2 -> -4 -> 12 -> 7 -> 10

Asked in Amazon Interview

Recommended Problem

Modify Linked List-1

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

[Solve Problem](https://practice.geeksforgeeks.org/problems/modify-linked-list-1/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 6.7K

**Approach:** The following steps are:

1. Split the list from the middle. Perform [**front and back split**](https://www.geeksforgeeks.org/merge-sort-for-linked-list/). If the number of elements is odd, the extra element should go in the 1st(front) list.
2. [**Reverse the 2nd(back) list**](https://www.geeksforgeeks.org/write-a-function-to-reverse-the-nodes-of-a-linked-list/).
3. Perform the required subtraction while traversing both list simultaneously.
4. Again reverse the 2nd list.
5. Concatenate the 2nd list back to the end of the 1st list.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 implementation to modify the contents

# of the linked list

# Linked list node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# Function to insert a node at the beginning

# of the linked list

**def** push(head\_ref, new\_data):

    # allocate node

    new\_node **=**Node(0)

    # put in the data

    new\_node.data **=** new\_data

    # link the old list at the end

    #of the new node

    new\_node.next **=** head\_ref

    # move the head to point to the new node

    head\_ref **=** new\_node

**return** head\_ref

front **=** None

back **=** None

# Split the nodes of the given list

# into front and back halves,

# and return the two lists

# using the reference parameters.

# Uses the fast/slow pointer strategy.

**def** frontAndBackSplit( head):

**global** front

**global** back

    slow **=** None

    fast **=** None

    slow **=** head

    fast **=** head.next

    # Advance 'fast' two nodes, and

    # advance 'slow' one node

**while** (fast !**=** None):

        fast **=** fast.next

**if** (fast !**=** None):

            slow **=** slow.next

            fast **=** fast.next

    # 'slow' is before the midpoint in the list,

    # so split it in two at that point.

    front **=** head

    back **=** slow.next

    slow.next **=** None

**return** head

# Function to reverse the linked list

**def** reverseList( head\_ref):

    current **=** None

    prev **=** None

    next **=** None

    current **=** head\_ref

    prev **=** None

**while** (current !**=** None):

        next **=** current.next

        current.next **=** prev

        prev **=** current

        current **=** next

    head\_ref **=** prev

**return** head\_ref

# perform the required subtraction operation

# on the 1st half of the linked list

**def** modifyTheContentsOf1stHalf():

**global** front

**global** back

    front1 **=** front

    back1 **=** back

    # traversing both the lists simultaneously

**while** (back1 !**=** None):

        # subtraction operation and node data

        # modification

        front1.data **=** front1.data **-** back1.data

        front1 **=** front1.next

        back1 **=** back1.next

# function to concatenate the 2nd(back) list

# at the end of the 1st(front) list and

# returns the head of the new list

**def** concatFrontAndBackList( front, back):

    head **=** front

**if**(front **==** None):

**return** back

**while** (front.next !**=** None):

        front **=** front.next

    front.next **=** back

**return** head

# function to modify the contents of the linked list

**def** modifyTheList( head):

**global** front

**global** back

    # if list is empty or contains only single node

**if** (head **==** None **or** head.next **==** None):

**return** head

    front **=** None

    back **=** None

    # split the list into two halves

    # front and back lists

    frontAndBackSplit(head)

    # reverse the 2nd(back) list

    back **=** reverseList(back)

    # modify the contents of 1st half

    modifyTheContentsOf1stHalf()

    # agains reverse the 2nd(back) list

    back **=** reverseList(back)

    # concatenating the 2nd list back to the

    # end of the 1st list

    head **=** concatFrontAndBackList(front, back)

    # pointer to the modified list

**return** head

# function to print the linked list

**def** printList( head):

**if** (head **==** None):

**return**

**while** (head.next !**=** None):

        print(head.data , " -> ",end**=**"")

        head **=** head.next

**print**(head.data )

# Driver Code

head **=** None

# creating the linked list

head **=** push(head, 10)

head **=** push(head, 7)

head **=** push(head, 12)

head **=** push(head, 8)

head **=** push(head, 9)

head **=** push(head, 2)

# modify the linked list

head **=** modifyTheList(head)

# print the modified linked list

**print**( "Modified List:" )

printList(head)

# This code is contributed by Arnab Kundu

**Output**

Modified List:  
-8 -> 2 -> -4 -> 12 -> 7 -> 10

**Time Complexity:** **O(n),**where **n** in the number of nodes.

**Space complexity**: **O(1)**since using constant space to modify pointers

**Another approach (Using Stack) :**

1. Find the starting point of second half Linked List.
2. Push all elements of second half list into stack s.
3. Traverse list starting from head using temp until stack is not empty and do Modify temp->data by subtracting the top element of stack for every node.

Below is the implementation using stack.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 implementation to modify the

# contents of the linked list

# Linked list node

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** 0

        self.next **=** None

# Function to insert a node at the

# beginning of the linked list

**def** append(head\_ref, new\_data):

    # Allocate node

    new\_node **=** Node()

    # Put in the data

    new\_node.data **=** new\_data

    # Link the old list at the end

    # of the new node

    new\_node.next **=** head\_ref

    # Move the head to point to the new node

    head\_ref **=** new\_node

**return** head\_ref

# Function to print the linked list

**def** printList(head):

**if** (**not** head):

**return**;

**while** (head.next !**=** None):

**print**(head.data, end **=** ' -> ')

        head **=** head.next

**print**(head.data)

# Function to middle node of list.

**def** find\_mid(head):

    temp **=** head

    slow **=** head

    fast **=** head

**while** (fast **and** fast.next):

        # Advance 'fast' two nodes, and

        # advance 'slow' one node

        slow **=** slow.next

        fast **=** fast.next.next

    # If number of nodes are odd then

    # update slow by slow.next;

**if** (fast):

        slow **=** slow.next

**return** slow

# Function to modify the contents of

# the linked list.

**def** modifyTheList(head, slow):

    # Create Stack.

    s **=** []

    temp **=** head

**while** (slow):

        s.append(slow.data)

        slow **=** slow.next

    # Traverse the list by using

    # temp until stack is empty.

**while** (len(s) !**=** 0):

        temp.data **=** temp.data **-** s[**-**1]

        temp **=** temp.next

        s.pop()

# Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    head **=** None

    # creating the linked list

    head **=** append(head, 10)

    head **=** append(head, 7)

    head **=** append(head, 12)

    head **=** append(head, 8)

    head **=** append(head, 9)

    head **=** append(head, 2)

    # Call Function to Find the

    # starting point of second half of list.

    mid **=** find\_mid(head)

    # Call function to modify the

    # contents of the linked list.

    modifyTheList( head, mid)

    # Print the modified linked list

**print**("Modified List:")

    printList(head)

# This code is contributed by rutvik\_56

**Output**

Modified List:  
-8 -> 2 -> -4 -> 12 -> 7 -> 10

**Time Complexity : O(n)**

**Space Complexity : O(n/2)**

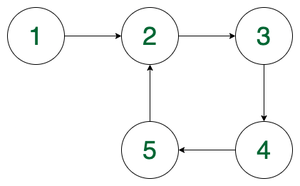
*From <*[*https://www.geeksforgeeks.org/modify-contents-linked-list/*](https://www.geeksforgeeks.org/modify-contents-linked-list/)*>*

**Hard Questions:**

**Detect loop in a linked list**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 14 Dec, 2022
* Read
* Discuss(118)
* Courses
* Practice
* Video

Given a linked list, check if the linked list has a loop or not. The below diagram shows a linked list with a loop.



*Linked List with Loop*

Recommended Problem

Detect Loop in linked list

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[two-pointer-algorithm](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=two-pointer-algorithm&sortBy=submissions)

+2 more

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Accolite&sortBy=submissions)

[Amazon](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Amazon&sortBy=submissions)

+16 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/detect-loop-in-linked-list/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 3.1L

**Detect loop in a linked list using**[**Hashing**](https://www.geeksforgeeks.org/introduction-to-hashing-data-structure-and-algorithm-tutorials/)**:**

*The idea is to insert the nodes in the****hashmap****and whenever a node is encountered that is already present in the hashmap then return true.*

Follow the steps below to solve the problem:

* Traverse the list individually and keep putting the node addresses in a Hash Table.
* At any point, if NULL is reached then return false
* If the next of the current nodes points to any of the previously stored nodes in  Hash then return true.

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to detect loop

# in the linked list

# Node class

**class** Node:

    # Constructor to initialize

    # the node object

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** LinkedList:

    # Function to initialize head

**def** \_\_init\_\_(self):

        self.head **=** None

    # Function to insert a new

    # node at the beginning

**def** push(self, new\_data):

        new\_node **=** Node(new\_data)

        new\_node.next **=** self.head

        self.head **=** new\_node

    # Utility function to print it

    # the linked LinkedList

**def** printList(self):

        temp **=** self.head

**while**(temp):

            print(temp.data, end**=**" ")

            temp **=** temp.next

**def** detectLoop(self):

        s **=** set()

        temp **=** self.head

**while** (temp):

            # If we already have

            # this node in hashmap it

            # means there is a cycle

            # (Because we are encountering

            # the node second time).

**if** (temp **in** s):

**return** True

            # If we are seeing the node for

            # the first time, insert it in hash

            s.add(temp)

            temp **=** temp.next

**return** False

# Driver program for testing

llist **=** LinkedList()

llist.push(20)

llist.push(4)

llist.push(15)

llist.push(10)

# Create a loop for testing

llist.head.next.next.next.next **=** llist.head

**if**(llist.detectLoop()):

**print**("Loop Found")

**else**:

**print**("No Loop ")

# This code is contributed by Gitanjali.

**Output**

Loop Found

**Time complexity:** O(N), Only one traversal of the loop is needed.

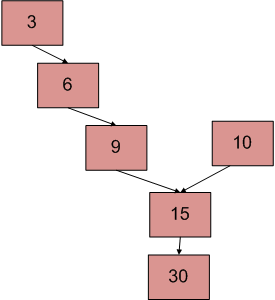
**Auxiliary Space:** O(N), N is the space required to store the value in the hashmap.

*From <*[*https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/*](https://www.geeksforgeeks.org/detect-loop-in-a-linked-list/)*>*

**Write a function to get the intersection point of two Linked Lists**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 16 Dec, 2022
* Read
* Discuss(472)
* Courses
* Practice
* Video

There are two singly linked lists in a system. By some programming error, the end node of one of the linked lists got linked to the second list, forming an inverted Y-shaped list. Write a program to get the point where two linked lists merge.



The above diagram shows an example with two linked lists having 15 as intersection points.

Recommended Problem

Intersection Point in Y Shapped Linked Lists

[Linked List](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Linked%20List&sortBy=submissions)

[Data Structures](https://practice.geeksforgeeks.org/explore?page=1&category%5b%5d=Data%20Structures&sortBy=submissions)

[Accolite](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Accolite&sortBy=submissions)

[Adobe](https://practice.geeksforgeeks.org/explore?page=1&company%5b%5d=Adobe&sortBy=submissions)

+12 more

[Solve Problem](https://practice.geeksforgeeks.org/problems/intersection-point-in-y-shapped-linked-lists/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article)

Submission count: 1.9L

**Method 1(Simply use two loops):**

Use 2 nested for loops. The outer loop will be for each node of the 1st list and the inner loop will be for the 2nd list. In the inner loop, check if any of the nodes of the 2nd list is the same as the current node of the first linked list. The time complexity of this method will be O(M \* N) where m and n are the numbers of nodes in two lists.

Below is the code for the above approach:

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python program to get intersection point of two linked list

# Link list node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# function to get the intersection point of two linked lists head1 and head

**def** getIntersectionNode(head1, head2):

**while** head2:

        temp **=** head1

**while** temp:

            # if both Nodes are same

**if** temp **==** head2:

**return** head2

            temp **=** temp.next

        head2 **=** head2.next

    # intersection is not present between the lists

**return** None

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    '''

    Create two linked lists

    1st 3->6->9->15->30

    2nd 10->15->30

    15 is the intersection point

    '''

    newNode **=** Node(10)

    head1 **=** newNode

    newNode **=** Node(3)

    head2 **=** newNode

    newNode **=** Node(6)

    head2.next **=** newNode

    newNode **=** Node(9)

    head2.next.next **=** newNode

    newNode **=** Node(15)

    head1.next **=** newNode

    head2.next.next.next **=** newNode

    newNode **=** Node(30)

    head1.next.next **=** newNode

    intersectionPoint **=** getIntersectionNode(head1, head2)

**if not** intersectionPoint:

        print(" No Intersection Point ")

**else**:

**print**("Intersection Point:", intersectionPoint.data)

# This code is contributed by Tapesh(tapeshdua420)

**Output**

Intersection Point: 15

**Time Complexity:**O(m\*n), where m and n are number of nodes in two linked list.

**Auxiliary Space:** O(1), Constant Space is used.

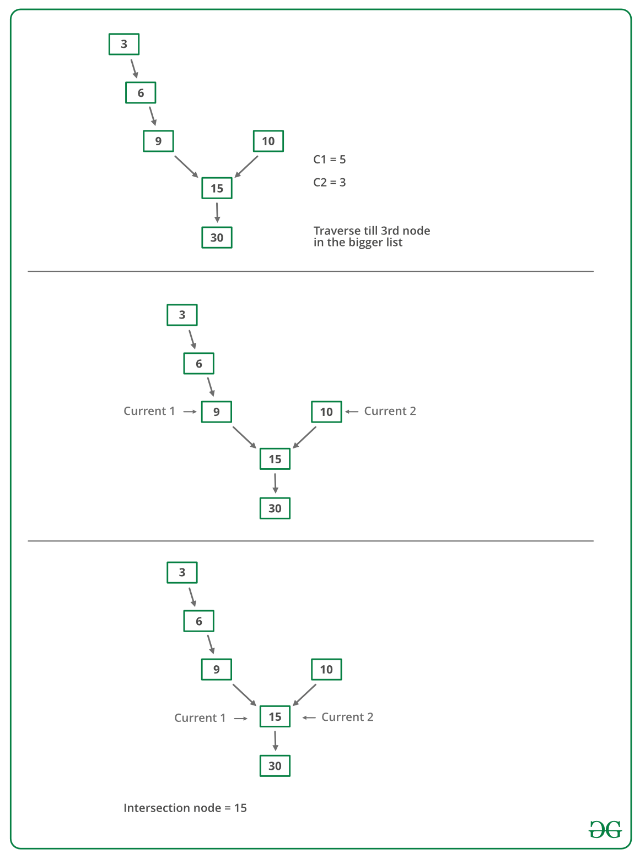
**Method 2 (Mark Visited Nodes):**

This solution requires modifications to the basic linked list data structure. Have a visited flag with each node. Traverse the first linked list and keep marking visited nodes. Now traverse the second linked list, If you see a visited node again then there is an intersection point, return the intersecting node. This solution works in **O(m+n)** but requires additional information with each node. A variation of this solution that doesn’t require modification to the basic data structure can be implemented using a hash. Traverse the first linked list and store the addresses of visited nodes in a hash. Now traverse the second linked list and if you see an address that already exists in the hash then return the intersecting node.

**Method 3(Using the difference in node counts)**

* Get the count of the nodes in the first list, let the count be c1.
* Get the count of the nodes in the second list, let the count be c2.
* Get the difference of counts **d = abs(c1 – c2)**
* Now traverse the bigger list from the first node to d nodes so that from here onwards both the lists have an equal no of nodes
* Then we can traverse both lists in parallel till we come across a common node. (Note that getting a common node is done by comparing the address of the nodes)

Below image is a dry run of the above approach:



Below is the implementation of the above approach :

* C++
* C
* Java
* Python3
* C#
* Javascript

# defining a node for LinkedList

**class** Node:

**def** \_\_init\_\_(self,data):

    self.data**=**data

    self.next**=**None

**def** getIntersectionNode(head1,head2):

  #finding the total number of elements in head1 LinkedList

    c1**=**getCount(head1)

  #finding the total number of elements in head2 LinkedList

    c2**=**getCount(head2)

  #Traverse the bigger node by 'd' so that from that node onwards, both LinkedList

  #would be having same number of nodes and we can traverse them together.

**if** c1 > c2:

        d**=**c1**-**c2

**return** \_getIntersectionNode(d,head1,head2)

**else**:

        d**=**c2**-**c1

**return** \_getIntersectionNode(d,head2,head1)

**def** \_getIntersectionNode(d,head1,head2):

    current1**=**head1

    current2**=**head2

**for** i **in** range(d):

**if** current1 **is** None:

**return -**1

        current1**=**current1.next

**while** current1 **is not** None **and** current2 **is not** None:

    # Instead of values, we need to check if there addresses are same

    # because there can be a case where value is same but that value is

    #not an intersecting point.

**if** current1 **is** current2:

**return** current1.data # or current2.data ( the value would be same)

        current1**=**current1.next

        current2**=**current2.next

  # Incase, we are not able to find our intersecting point.

**return -**1

#Function to get the count of a LinkedList

**def** getCount(node):

    cur**=**node

    count**=**0

**while** cur **is not** None:

        count**+=**1

        cur**=**cur.next

**return** count

**if** \_\_name\_\_ **==** '\_\_main\_\_':

  # Creating two LinkedList

  # 1st one: 3->6->9->15->30

  # 2nd one: 10->15->30

  # We can see that 15 would be our intersection point

  # Defining the common node

  common**=**Node(15)

  #Defining first LinkedList

  head1**=**Node(3)

  head1.next**=**Node(6)

  head1.next.next**=**Node(9)

  head1.next.next.next**=**common

  head1.next.next.next.next**=**Node(30)

  # Defining second LinkedList

  head2**=**Node(10)

  head2.next**=**common

  head2.next.next**=**Node(30)

**print**("The node of intersection is ",getIntersectionNode(head1,head2))

  # The code is contributed by Ansh Gupta.

**Output**

The node of intersection is 15

*From <*[*https://www.geeksforgeeks.org/write-a-function-to-get-the-intersection-point-of-two-linked-lists/*](https://www.geeksforgeeks.org/write-a-function-to-get-the-intersection-point-of-two-linked-lists/)*>*

**Circular Linked List Implementation of Circular Queue**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 04 Nov, 2022
* Read
* Discuss(8)
* Courses
* Practice
* Video

Prerequisite – [Circular Singly Linked List](https://www.geeksforgeeks.org/circular-singly-linked-list-insertion/) We have discussed basics and how to implement circular queue using array in set 1. [Circular Queue | Set 1 (Introduction and Array Implementation)](https://www.geeksforgeeks.org/circular-queue-set-1-introduction-array-implementation/) In this post another method of circular queue implementation is discussed, using Circular Singly Linked List.

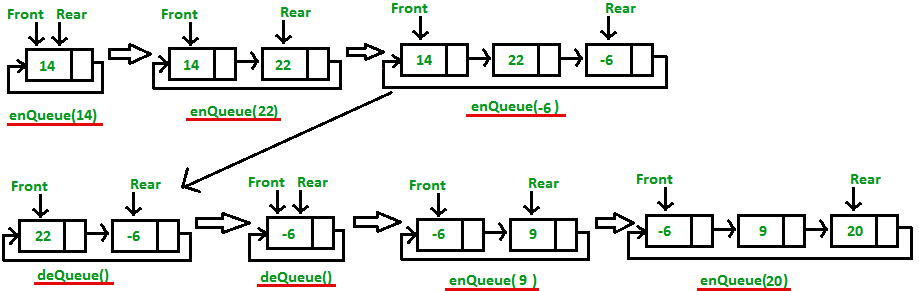
*Operations on Circular Queue:*

* ***Front:****Get the front item from queue.*
* ***Rear:****Get the last item from queue.*
* ***enQueue(value)****This function is used to insert an element into the circular queue. In a circular queue, the new element is always inserted at Rear position.*

1. *Create a new node dynamically and insert value into it.*
2. *Check if front==NULL, if it is true then front = rear = (newly created node)*
3. *If it is false then rear=(newly created node) and rear node always contains the address of the front node.*

* ***deQueue()****This function is used to delete an element from the circular queue. In a queue, the element is always deleted from front position.*

1. *Check whether queue is empty or not means front == NULL.*
2. *If it is empty then display Queue is empty. If queue is not empty then step 3*
3. *Check if (front==rear) if it is true then set front = rear = NULL else move the front forward in queue, update address of front in rear node and return the element.*

[](https://media.geeksforgeeks.org/wp-content/uploads/Operations-on-Circular-Queue.png)

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

Below is the implementation of the above approach:

* C++
* Java
* Python3
* C#

# Python3 program for insertion and

# deletion in Circular Queue

# Structure of a Node

**class** Node:

**def** \_\_init\_\_(self):

        self.data **=** None

        self.link **=** None

**class** Queue:

**def** \_\_init\_\_(self):

        front **=** None

        rear **=** None

# Function to create Circular queue

**def** enQueue(q, value):

    temp **=** Node()

    temp.data **=** value

**if** (q.front **==** None):

        q.front **=** temp

**else**:

        q.rear.link **=** temp

    q.rear **=** temp

    q.rear.link **=** q.front

# Function to delete element from

# Circular Queue

**def** deQueue(q):

**if** (q.front **==** None):

        print("Queue is empty")

**return -**999999999999

    # If this is the last node to be deleted

    value **=** None # Value to be dequeued

**if** (q.front **==** q.rear):

        value **=** q.front.data

        q.front **=** None

        q.rear **=** None

**else**: # There are more than one nodes

        temp **=** q.front

        value **=** temp.data

        q.front **=** q.front.link

        q.rear.link **=** q.front

**return** value

# Function displaying the elements

# of Circular Queue

**def** displayQueue(q):

    temp **=** q.front

    print("Elements in Circular Queue are: ",

                                   end **=** " ")

**while** (temp.link !**=** q.front):

        print(temp.data, end **=** " ")

        temp **=** temp.link

**print**(temp.data)

# Driver Code

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    # Create a queue and initialize

    # front and rear

    q **=** Queue()

    q.front **=** q.rear **=** None

    # Inserting elements in Circular Queue

    enQueue(q, 14)

    enQueue(q, 22)

    enQueue(q, 6)

    # Display elements present in

    # Circular Queue

    displayQueue(q)

    # Deleting elements from Circular Queue

**print**("Deleted value = ", deQueue(q))

    print("Deleted value = ", deQueue(q))

    # Remaining elements in Circular Queue

    displayQueue(q)

    enQueue(q, 9)

    enQueue(q, 20)

    displayQueue(q)

# This code is contributed by PranchalK

**Output**

Elements in Circular Queue are: 14 22 6  
Deleted value = 14  
Deleted value = 22  
Elements in Circular Queue are: 6  
Elements in Circular Queue are: 6 9 20

*From <*[*https://www.geeksforgeeks.org/circular-linked-list-implementation-of-circular-queue/*](https://www.geeksforgeeks.org/circular-linked-list-implementation-of-circular-queue/)*>*

**Josephus Circle implementation using STL list**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 26 Jul, 2022
* Read
* Discuss(15)
* Courses
* Practice
* Video

There are n people standing in a circle waiting to be executed. The counting out begins at some point in the circle and proceeds around the circle in a fixed direction. In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom. Given the total number of person n and a number k which indicates that k-1 persons are skipped and the kth person is killed in the circle. The task is to choose the place in the initial circle so that you are the last one remaining and so survive. (0-based indexing) .

**Examples :**

Input : Length of circle : n = 4  
 Count to choose next : k = 2  
Output : 0

Input : n = 5  
 k = 3  
Output : 3

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have already discussed different solutions to this problem ([here](https://www.geeksforgeeks.org/josephus-problem-set-1-a-on-solution/), [here,](https://www.geeksforgeeks.org/?p=158048)and [here](https://www.geeksforgeeks.org/josephus-problem-using-bit-magic/)). In this post, a  C++ STL-based solution using a list container is discussed which uses the idea of a circular list.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find last man standing

# /\* structure for a node in circular

#    linked list \*/

**class** Node:

**def** \_\_init\_\_(self, x):

        self.data **=** x

        self.next **=** None

# /\* Function to find the only person left

#    after one in every m-th node is killed

#    in a circle of n nodes \*/

**def** getJosephusPosition(m, n):

    # Create a circular linked list of

    # size N.

    head **=** Node(1)

    prev **=** head

**for** i **in** range(2, n **+** 1):

        prev.next **=** Node(i)

        prev **=** prev.next

    prev.next **=** head # Connect last

                       #node to first

    #/\* while only one node is left in the

    #linked list\*/

    ptr1 **=** head

    ptr2 **=** head

**while** (ptr1.next !**=** ptr1):

        # Find m-th node

        count **=** 1

**while** (count !**=** m):

            ptr2 **=** ptr1

            ptr1 **=** ptr1.next

            count **+=** 1

        # /\* Remove the m-th node \*/

        ptr2.next **=** ptr1.next

        # free(ptr1)

        ptr1 **=** ptr2.next

**print**("Last person left standing (Josephus Position) is ", ptr1.data)

# /\* Driver program to test above functions \*/

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    n **=** 14

    m **=** 2

    getJosephusPosition(m, n)

# This code is contributed by mohit kumar 29

**Output**

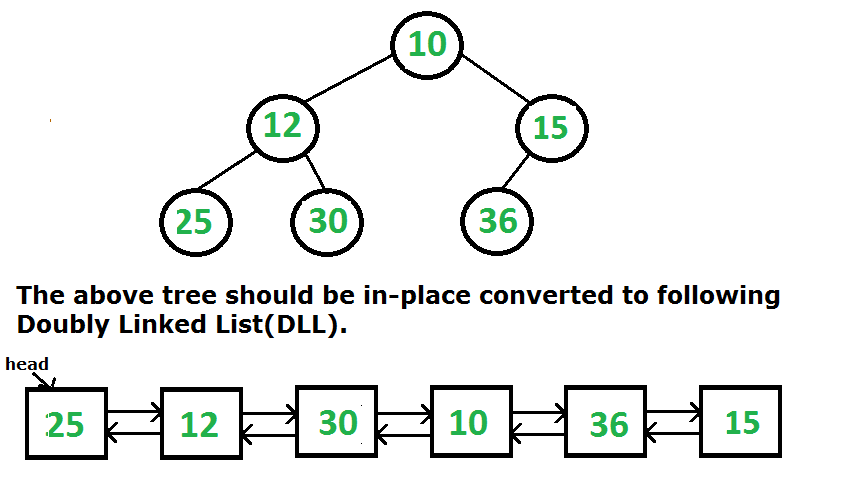
12

*From <*[*https://www.geeksforgeeks.org/josephus-circle-using-circular-linked-list/*](https://www.geeksforgeeks.org/josephus-circle-using-circular-linked-list/)*>*

**Convert given Binary Tree to Doubly Linked List in Linear time**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 04 Nov, 2022
* Read
* Discuss(35)
* Courses
* Practice
* Video

Given a Binary Tree (BT), convert it to a Doubly Linked List(DLL) In-Place. The left and right pointers in nodes are to be used as previous and next pointers respectively in converted DLL. The order of nodes in DLL must be same as Inorder of the given Binary Tree. The first node of Inorder traversal (left most node in BT) must be head node of the DLL.



[Recommended: Please solve it on “***PRACTICE***” first, before moving on to the solution.](https://practice.geeksforgeeks.org/problems/binary-tree-to-dll/1)

Below three different solutions have been discussed for this problem.

[Convert a given Binary Tree to Doubly Linked List | Set 1](https://www.geeksforgeeks.org/in-place-convert-a-given-binary-tree-to-doubly-linked-list/)

[Convert a given Binary Tree to Doubly Linked List | Set 2](https://www.geeksforgeeks.org/convert-a-given-binary-tree-to-doubly-linked-list-set-2/)

[Convert a given Binary Tree to Doubly Linked List | Set 3](https://www.geeksforgeeks.org/convert-given-binary-tree-doubly-linked-list-set-3/)

In the following implementation, we traverse the tree in inorder fashion. We add nodes at the beginning of current linked list and update head of the list using pointer to head pointer. Since we insert at the beginning, we need to process leaves in reverse order. For reverse order, we first traverse the right subtree before the left subtree. i.e. do a reverse inorder traversal.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to convert a given Binary Tree to Doubly Linked List

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.left **=** self.right **=** None

**class** BinaryTree:

    # A simple recursive function to convert a given

    # Binary tree to Doubly Linked List

    # root    --> Root of Binary Tree

    # head --> Pointer to head node of created doubly linked list

    root, head **=** None, None

**def** BToDll(self, root: Node):

**if** root **is** None:

**return**

        # Recursively convert right subtree

        self.BToDll(root.right)

        # Insert root into doubly linked list

        root.right **=** self.head

        # Change left pointer of previous head

**if** self.head **is not** None:

            self.head.left **=** root

        # Change head of doubly linked list

        self.head **=** root

        # Recursively convert left subtree

        self.BToDll(root.left)

    @staticmethod

**def** print\_list(head: Node):

**print**('Extracted Double Linked list is:')

**while** head **is not** None:

**print**(head.data, end **=** ' ')

            head **=** head.right

# Driver program to test above function

**if** \_\_name\_\_ **==** '\_\_main\_\_':

    """

    Constructing below tree

            5

        // \\

        3 6

        // \\ \\

        1 4 8

    // \\ // \\

    0 2 7 9

    """

    tree **=** BinaryTree()

    tree.root **=** Node(5)

    tree.root.left **=** Node(3)

    tree.root.right **=** Node(6)

    tree.root.left.left **=** Node(1)

    tree.root.left.right **=** Node(4)

    tree.root.right.right **=** Node(8)

    tree.root.left.left.left **=** Node(0)

    tree.root.left.left.right **=** Node(2)

    tree.root.right.right.left **=** Node(7)

    tree.root.right.right.right **=** Node(9)

    tree.BToDll(tree.root)

    tree.print\_list(tree.head)

# This code is contributed by Rajat Srivastava

**Output**

Extracted Double Linked list is:  
0 1 2 3 4 5 6 7 8 9

**Time Complexity:** **O(n),** as the solution does a single traversal of given Binary Tree.

**Auxiliary Space: O(n)**

*From <*[*https://www.geeksforgeeks.org/convert-given-binary-tree-to-doubly-linked-list-in-linear-time/*](https://www.geeksforgeeks.org/convert-given-binary-tree-to-doubly-linked-list-in-linear-time/)*>*

**Priority Queue using Doubly Linked List**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 06 Oct, 2022
* Read
* Discuss(9)
* Courses
* Practice
* Video

Given Nodes with their priority, implement a priority queue using doubly linked list.

**Prerequisite :** [Priority Queue](https://www.geeksforgeeks.org/priority-queue-set-1-introduction/)

* push(): This function is used to insert a new data into the queue.
* pop(): This function removes the element with the lowest priority value from the queue.
* peek() / top(): This function is used to get the lowest priority element in the queue without removing it from the queue.

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Approach :**

1. Create a doubly linked list having fields info(hold the information of the Node), priority(hold the priority of the Node), prev(point to previous Node), next(point to next Node).

2. Insert the element and priority in the Node.

3. Arrange the Nodes in the increasing order of priority.

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

Below is the implementation of above steps :

* C++
* C
* Java
* Python3
* C#
* Javascript

# Python3 code to implement priority

# queue using doubly linked list

# Linked List Node

**class** Node:

**def** \_\_init\_\_(self):

        self.info **=** 0

        self.priority **=** 0

        self.next **=** None

        self.prev **=** None

front **=** None

rear **=** None

# Function to insert a new Node

**def** push(fr, rr, n, p):

**global** front, rear

    news **=** Node()

    news.info **=** n

    news.priority **=** p

    # If linked list is empty

**if** (fr **==** None):

        fr **=** news

        rr **=** news

        news.next **=** None

**else**:

        # If p is less than or equal fr

        # node's priority, then insert at

        # the fr.

**if** (p <**=** (fr).priority):

            news.next **=** fr

            (fr).prev **=** news.next

            fr **=** news

        # If p is more rr node's priority,

        # then insert after the rr.

**elif** (p > (rr).priority):

            news.next **=** None

            (rr).next **=** news

            news.prev **=** (rr).next

            rr **=** news

        # Handle other cases

**else**:

            # Find position where we need to

            # insert.

            start **=** (fr).next

**while** (start.priority > p):

                start **=** start.next

            (start.prev).next **=** news

            news.next **=** start.prev

            news.prev **=** (start.prev).next

            start.prev **=** news.next

    front **=** fr

    rear **=** rr

# Return the value at rr

**def** peek(fr):

**return** fr.info

**def** isEmpty(fr):

**return** fr **==** None

# Removes the element with the

# least priority value from the list

**def** pop(fr, rr):

**global** front , rear

    temp **=** fr

    res **=** temp.info

    (fr) **=** (fr).next

**if** (fr **==** None):

        rr **=** None

    front **=** fr

    rear **=** rr

**return** res

# Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    push( front, rear, 2, 3)

    push( front, rear, 3, 4)

    push( front, rear, 4, 5)

    push( front, rear, 5, 6)

    push( front, rear, 6, 7)

    push( front, rear, 1, 2)

    print(pop(front, rear))

**print**(peek(front))

# This code is contributed by rutvik\_56

**Output:**

1  
2

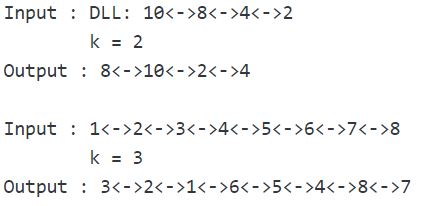
*From <*[*https://www.geeksforgeeks.org/priority-queue-using-doubly-linked-list/*](https://www.geeksforgeeks.org/priority-queue-using-doubly-linked-list/)*>*

**Reverse a doubly linked list in groups of given size**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 28 Sep, 2022
* Read
* Discuss(13)
* Courses
* Practice
* Video

Given a doubly linked list containing **n** nodes. The problem is to reverse every group of **k** nodes in the list.

**Examples:**



[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

**Prerequisite:** [Reverse a doubly linked list | Set-2.](https://www.geeksforgeeks.org/reverse-doubly-linked-list-set-2/)

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

**Approach:** Create a recursive function say **reverse(head, k)**. This function receives the head or the first node of each group of **k** nodes. It reverses those groups of **k** nodes by applying the approach discussed in [Reverse a doubly linked list | Set-2.](https://www.geeksforgeeks.org/reverse-doubly-linked-list-set-2/) After reversing the group of **k** nodes the function checks whether next group of nodes exists in the list or not. If a group exists then it makes a recursive call to itself with the first node of the next group and makes the necessary adjustments with the next and previous links of that group. Finally, it returns the new head node of the reversed group.

* C++
* Java
* Python3
* C#
* Javascript

# Python implementation to reverse a doubly linked list

# in groups of given size

# Link list node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** next

# function to get a new node

**def** getNode(data):

    # allocate space

    new\_node **=** Node(0)

    # put in the data

    new\_node.data **=** data

    new\_node.next **=** new\_node.prev **=** None

**return** new\_node

# function to insert a node at the beginning

# of the Doubly Linked List

**def** push(head\_ref, new\_node):

    # since we are adding at the beginning,

    # prev is always None

    new\_node.prev **=** None

    # link the old list off the new node

    new\_node.next **=** (head\_ref)

    # change prev of head node to new node

**if** ((head\_ref) !**=** None):

        (head\_ref).prev **=** new\_node

    # move the head to point to the new node

    (head\_ref) **=** new\_node

**return** head\_ref

# function to reverse a doubly linked list

# in groups of given size

**def** revListInGroupOfGivenSize( head, k):

    current **=** head

    next **=** None

    newHead **=** None

    count **=** 0

    # reversing the current group of k

    # or less than k nodes by adding

    # them at the beginning of list

    # 'newHead'

**while** (current !**=** None **and** count < k):

        next **=** current.next

        newHead **=** push(newHead, current)

        current **=** next

        count **=** count **+** 1

    # if next group exists then making the desired

    # adjustments in the link

**if** (next !**=** None):

        head.next **=** revListInGroupOfGivenSize(next, k)

        head.next.prev **=** head

    # pointer to the new head of the

    # reversed group

**return** newHead

# Function to print nodes in a

# given doubly linked list

**def** printList(head):

**while** (head !**=** None):

**print**( head.data , end**=**" ")

        head **=** head.next

# Driver program to test above

# Start with the empty list

head **=** None

# Create doubly linked: 10<.8<.4<.2

head **=** push(head, getNode(2))

head **=** push(head, getNode(4))

head **=** push(head, getNode(8))

head **=** push(head, getNode(10))

k **=** 2

print("Original list: ")

printList(head)

# Reverse doubly linked list in groups of

# size 'k'

head **=** revListInGroupOfGivenSize(head, k)

**print**("\nModified list: ")

printList(head)

# This code is contributed by Arnab Kundu

**Output**

Original list: 10 8 4 2   
Modified list: 8 10 2 4

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

*From <*[*https://www.geeksforgeeks.org/reverse-doubly-linked-list-groups-given-size/*](https://www.geeksforgeeks.org/reverse-doubly-linked-list-groups-given-size/)*>*

**Reverse a stack without using extra space in O(n)**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 10 Aug, 2022
* Read
* Discuss(24)
* Courses
* Practice
* Video

Reverse a [Stack](https://www.geeksforgeeks.org/stack-data-structure/) without using recursion and extra space. Even the functional Stack is not allowed.

**Examples:**

Input : 1->2->3->4  
Output : 4->3->2->1

Input : 6->5->4  
Output : 4->5->6

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

We have discussed a way of reversing a stack in the below post.

[Reverse a Stack using Recursion](https://www.geeksforgeeks.org/reverse-a-stack-using-recursion/)

[](https://practice.geeksforgeeks.org/courses/complete-interview-preparation?utm_source=article&utm_medium=article&utm_campaign=complete-interview-preparation)

The above solution requires O(n) extra space. We can reverse a stack in O(1) time if we internally represent the stack as a linked list. Reverse a stack would require a reversing of a linked list which can be done with O(n) time and O(1) extra space.

Note that push() and pop() operations still take O(1) time.

**Implementation:**

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to implement Stack

# using linked list so that reverse

# can be done with O(1) extra space.

**class** StackNode:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

**class** Stack:

**def** \_\_init\_\_(self):

        self.top **=** None

    # Push and pop operations

**def** push(self, data):

**if** (self.top **==** None):

            self.top **=** StackNode(data)

**return**

        s **=** StackNode(data)

        s.next **=** self.top

        self.top **=** s

**def** pop(self):

        s **=** self.top

        self.top **=** self.top.next

**return** s

    # Prints contents of stack

**def** display(self):

        s **=** self.top

**while** (s !**=** None):

**print**(s.data, end **=** ' ')

            s **=** s.next

    # Reverses the stack using simple

    # linked list reversal logic.

**def** reverse(self):

        prev **=** self.top

        cur **=** self.top

        cur **=** cur.next

        succ **=** None

        prev.next **=** None

**while** (cur !**=** None):

            succ **=** cur.next

            cur.next **=** prev

            prev **=** cur

            cur **=** succ

        self.top **=** prev

# Driver code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    s **=** Stack()

    s.push(1)

    s.push(2)

    s.push(3)

    s.push(4)

    print("Original Stack")

    s.display()

**print**()

    # Reverse

    s.reverse()

    print("Reversed Stack")

    s.display()

# This code is contributed by rutvik\_56

**Output**

Original Stack  
4 3 2 1

Reversed Stack  
1 2 3 4

**Time Complexity: O(n)**, as we are using a loop to traverse n times. Where n is the number of nodes in the linked list.

**Auxiliary Space: O(1),** as we are not using any extra space.

*From <*[*https://www.geeksforgeeks.org/reverse-stack-without-using-extra-space/*](https://www.geeksforgeeks.org/reverse-stack-without-using-extra-space/)*>*

**Sublist Search (Search a linked list in another list)**

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)
* Last Updated : 20 Oct, 2022
* Read
* Discuss(30)
* Courses
* Practice
* Video

Given two linked lists, the task is to check whether the first list is present in 2nd list or not.

**Examples:**

***Input:****list1 =  10->20*

*list2  = 5->10->20*

***Output :****LIST FOUND*

***Input:****list1 =  1->2->3->4*

*list2  = 1->2->1->2->3->4*

***Output:****LIST FOUND*

***Input:****list1 =  1->2->3->4*

*list2  = 1->2->2->1->2->3*

***Output:****LIST NOT FOUND*

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

***Algorithm:***

1. *Take first node of second list.*
2. *Start matching the first list from this first node.*
3. *If whole lists match return true.*
4. *Else break and take first list to the first node again.*
5. *And take second list to its second node.*
6. *Repeat these steps until any of linked lists becomes empty.*
7. *If first list becomes empty then list found else not.*

Below is the implementation.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to find a list in second list

**class** Node:

**def** \_\_init\_\_(self, value **=** 0):

        self.value **=** value

        self.next **=** None

# Returns true if first list is

# present in second list

**def** findList(first, second):

    # If both linked lists are empty/None,

    # return True

**if not** first **and not** second:

**return** True

    # If ONLY one of them is empty,

    # return False

**if not** first **or not** second:

**return** False

    ptr1 **=** first

    ptr2 **=** second

    # Traverse the second LL by

    # picking nodes one by one

**while** ptr2:

        # Initialize 'ptr2' with current

        # node of 'second'

        ptr2 **=** second

        # Start matching first LL

        # with second LL

**while** ptr1:

            # If second LL become empty and

            # first not, return False,

            # since first LL has not been

            # traversed completely

**if not** ptr2:

**return** False

            # If value of both nodes from both

            # LLs are equal, increment pointers

            # for both LLs so that next value

            # can be matched

**else if** ptr1.value **==** ptr2.value:

                ptr1 **=** ptr1.next

                ptr2 **=** ptr2.next

            # If a single mismatch is found

            # OR ptr1 is None/empty,break out

            # of the while loop and do some checks

**else**:

**break**

        # check 1 :

        # If 'ptr1' is None/empty,that means

        # the 'first LL' has been completely

        # traversed and matched so return True

**if not** ptr1:

**return** True

        # If check 1 fails, that means, some

        # items for 'first' LL are still yet

        # to be matched, so start again by

        # bringing back the 'ptr1' to point

        # to 1st node of 'first' LL

        ptr1 **=** first

        # And increment second node element to next

        second **=** second.next

**return** False

# Driver Code

# Let us create two linked lists to

# test the above functions.

# Created lists would be be

# node\_a: 1->2->3->4

# node\_b: 1->2->1->2->3->4

node\_a **=** Node(1)

node\_a.next **=** Node(2)

node\_a.next.next **=** Node(3)

node\_a.next.next.next **=** Node(4)

node\_b **=** Node(1)

node\_b.next **=** Node(2)

node\_b.next.next **=** Node(1)

node\_b.next.next.next **=** Node(2)

node\_b.next.next.next.next **=** Node(3)

node\_b.next.next.next.next.next **=** Node(4)

**if** findList(node\_a, node\_b):

**print**("LIST FOUND")

**else**:

    print("LIST NOT FOUND")

# This code is contributed by GauriShankarBadola

**Output**

LIST FOUND

*From <*[*https://www.geeksforgeeks.org/sublist-search-search-a-linked-list-in-another-list/*](https://www.geeksforgeeks.org/sublist-search-search-a-linked-list-in-another-list/)*>*

**Partitioning a linked list around a given value and If we don’t care about making the elements of the list “stable”**

* Difficulty Level : [Medium](https://www.geeksforgeeks.org/medium/)
* Last Updated : 12 Jul, 2022
* Read
* Discuss(4)
* Courses
* Practice
* Video

Given a linked list and a value x, partition a linked list around a value x, such that all nodes less than x come before all nodes greater than or equal to x. If x is contained within the list the values of x only need to be after the elements less than x (see below). The partition element x can appear anywhere in the “right partition”; it does not need to appear between the left and right partitions.

Similar problem: [Partitioning a linked list around a given value and keeping the original order](https://www.geeksforgeeks.org/partitioning-a-linked-list-around-a-given-value-and-keeping-the-original-order/)

**Examples:**

Input : 3 -> 5 -> 10 -> 2 -> 8 -> 2 -> 1   
 x = 5  
Output : 1-> 2-> 2-> 3-> 5-> 10-> 8

[Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.](https://ide.geeksforgeeks.org/)

If we don’t care about making the elements of the list “stable” then we can instead rearrange the elements by growing the list at the head and tail.

In this approach, we start a “new” list (using the existing nodes). Elements bigger than the pivot element are put at the tail and elements smaller are put at the head. Each time we insert an element, we update either the head or tail.

Below is the implementation of above idea.

* C++
* Java
* Python3
* C#
* Javascript

# Python3 program to partition a

# linked list around a given value.

**import** math

# Link list Node

**class** Node:

**def** \_\_init\_\_(self, data):

        self.data **=** data

        self.next **=** None

# A utility function to create a new node

**def** newNode(data):

    new\_node **=** Node(data)

    new\_node.data **=** data

    new\_node.next **=** None

**return** new\_node

# Function to make a new list

# (using the existing nodes)

# and return head of new list.

**def** partition(head, x):

    # Let us initialize start and

    # tail nodes of new list

    tail **=** head

    # Now iterate original list

    # and connect nodes

    curr **=** head

**while** (curr !**=** None):

        next **=** curr.next

**if** (curr.data < x):

            # Insert node at head.

            curr.next **=** head

            head **=** curr

**else**:

            # Append to the list of greater values

            # Insert node at tail.

            tail.next **=** curr

            tail **=** curr

        curr **=** next

    tail.next **=** None

    # The head has changed, so we need

    # to return it to the user.

**return** head

# Function to print linked list

**def** printList(head):

    temp **=** head

**while** (temp !**=** None):

        print(temp.data, end **=** " ")

        temp **=** temp.next

# Driver Code

**if** \_\_name\_\_**==**'\_\_main\_\_':

    # Start with the empty list

    head **=** newNode(3)

    head.next **=** newNode(5)

    head.next.next **=** newNode(8)

    head.next.next.next **=** newNode(2)

    head.next.next.next.next **=** newNode(10)

    head.next.next.next.next.next **=** newNode(2)

    head.next.next.next.next.next.next **=** newNode(1)

    x **=** 5

    head **=** partition(head, x)

    printList(head)

# This code is contributed by AbhiThakur

**Output**

1 2 2 3 5 8 10

**Complexity Analysis:**

* **Time Complexity:** **O(n).**
* **Space Complexity: O(1),** as we are not using more than 4 pointers.

*From <*[*https://www.geeksforgeeks.org/partitioning-linked-list-around-given-value-dont-care-making-elements-list-stable/*](https://www.geeksforgeeks.org/partitioning-linked-list-around-given-value-dont-care-making-elements-list-stable/)*>*