Add Two Numbers in a Linked List - GeeksforGeeks

Source: https://www.geeksforgeeks.org/add-two-numbers-represented-by-linked-lists/

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Add Two Numbers in a Linked List Last Updated : 30 Aug, 2025 Given two non-negative integers represented as linked lists with heads head1 and head2, where each node contains a single digit, return a new linked list representing their sum. Note: The input lists may contain leading zeros, but the resulting sum list must not contain any leading zeros. Examples : Input: Explanation: Sum of 123 and 999 is 1122. Input: Output: 7 -> 0 Explanation: Sum of 63 and 7 is 70. Try it on GfG Practice Table of Content [Naive Approach] By creating a new list - O(m + n) Time and O(max(m, n)) Space [Expected Approach] By storing sum on the longer list - O(m + n) Time and O(1) Space [Other Approach] Using Recursion - O(m + n) Time and O(max(m, n)) Space [Naive Approach] By creating a new list - O(m + n) Time and O(max(m, n)) Space To sum two linked lists, start by creating an empty linked list, say result, for the sum. Reverse both original linked lists to start from the least significant digit. Use two pointers to traverse the reversed lists simultaneously. For each pair of nodes, compute their sum and if the sum exceeds 9, store the remainder (sum modulo 10) in the new node and forward the carry to the next pair of nodes. Append each new node to result. Continue until both lists are fully traversed, handling any remaining nodes from the longer list and carrying over any final carry. Finally, reverse the result linked list to get the sum of the two linked list. C++ #include <iostream> using namespace std ; class Node { public : int data ; Node * next ; Node (int val) { data = val ; next = nullptr ; } }; // Function to reverse the linked list Node * reverse (Node * head) { Node * prev = nullptr , * curr = head , * next ; while (curr != nullptr) { next = curr -> next ; curr -> next = prev ; prev = curr ; curr = next ; } return prev ; } // Function to trim leading zeros in linked list Node * trimLeadingZeros (Node * head) { while (head -> next != nullptr && head -> data == 0) head = head -> next ; return head ; } // Function to add two numbers represented by linked list Node * addTwoLists (Node * num1 , Node * num2) { Node * res = nullptr , * curr = nullptr ; int carry = 0 ; num1 = trimLeadingZeros (num1); num2 = trimLeadingZeros (num2); num1 = reverse (num1); num2 = reverse (num2); // Iterate till either num1 is not empty or num2 is // not empty or carry is greater than 0 while (num1 != nullptr || num2 != nullptr || carry != 0) { int sum = carry ; if (num1 != nullptr) { sum += num1 -> data ; num1 = num1 -> next ; } if (num2 != nullptr) { sum += num2 -> data ; num2 = num2 -> next ; } Node * newNode = new Node (sum % 10); carry = sum / 10 ; // If this is the first node, then make this node // as the head of the resultant linked list if (res == nullptr) { res = newNode ; curr = newNode ; } else { // Append new node to resultant linked list // and move to the next node curr -> next = newNode ; curr = curr -> next ; } } return reverse (res); } void printList (Node * head) { Node * curr = head ; while (curr != nullptr) { cout << curr -> data ; if (curr -> next != NULL){ cout << " -> " ; } curr = curr -> next ; } cout << " \n " ; } int main () { Node * num1 = new Node (1); num1 -> next = new Node (2); num1 -> next -> next = new Node (3); Node * num2 = new Node (9); num2 -> next = new Node (9); num2 -> next -> next = new Node (9); Node * sum = addTwoLists (num1 , num2); printList (sum); return 0 ; } C #include <stdio.h> struct Node { int data ; struct Node * next ; }; struct Node * createNode (int val); // Function to reverse the linked list struct Node * reverse (struct Node * head) { struct Node * prev = NULL , * curr = head , * next ; while (curr != NULL) { next = curr -> next ; curr -> next = prev ; prev = curr ; curr = next ; } return prev ; } // Function to trim leading zeros in linked list struct Node * trimLeadingZeros (struct Node * head) { while (head -> next != NULL && head -> data == 0) head = head -> next ; return head ; } // Function to add two numbers represented by linked list struct Node * addTwoLists (struct Node * num1 , struct Node * num2) { struct Node * res = NULL , * curr = NULL ; int carry = 0 ; num1 = trimLeadingZeros (num1);

num2 = trimLeadingZeros (num2); num1 = reverse (num1); num2 = reverse (num2); // Iterate till either num1 is not empty or num2 is // not empty or carry is greater than 0 while (num1 != NULL || num2 != NULL || carry != 0) { int sum = carry ; if (num1 != NULL) { sum += num1 -> data ; num1 = num1 -> next ; } if (num2 != NULL) { sum += num2 -> data ; num2 = num2 -> next ; } struct Node * newNode = createNode (sum % 10); carry = sum / 10 ; // If this is the first node, then make this node // as the head of the resultant linked list if (res == NULL) { res = newNode ; curr = newNode ; } else { // Append new node to resultant linked list // and move to the next node curr -> next = newNode ; curr = curr -> next ; } } return reverse (res); } void printList (struct Node * head) { struct Node * curr = head ; while (curr != NULL) { printf ("%d" , curr -> data); if (curr -> next != NULL){ printf (" -> "); } curr = curr -> next ; } printf (" \n "); } struct Node * createNode (int val) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = val ; node -> next = NULL ; return node ; } int main () { struct Node * num1 = createNode (1); num1 -> next = createNode (2); num1 -> next -> next = createNode (3); struct Node * num2 = createNode (9); num2 -> next = createNode (9); num2 -> next -> next = createNode (9); struct Node * sum = addTwoLists (num1 , num2); printList (sum); return 0 ; } Java class Node { int data ; Node next ; Node (int val) { data = val ; next = null ; } } class GfG { // Function to reverse the linked list static Node reverse (Node head) { Node prev = null ; Node curr = head ; Node next ; while (curr != null) { next = curr . next ; curr . next = prev ; prev = curr ; curr = next ; } return prev ; } // Function to trim leading zeros in linked list static Node trimLeadingZeros (Node head) { while (head != null && head . data == 0) { head = head . next ; } return head ; } // Function to add two numbers represented by linked list static Node addTwoLists (Node num1 , Node num2) { Node res = null ; Node curr = null ; int carry = 0 ; num1 = trimLeadingZeros (num1); num2 = trimLeadingZeros (num2); num1 = reverse (num1); num2 = reverse (num2); // Iterate till either num1 is not empty or num2 is // not empty or carry is greater than 0 while (num1 != null || num2 != null || carry != 0) { int sum = carry ; if (num1 != null) { sum += num1 . data ; num1 = num1 . next ; } if (num2 != null) { sum += num2 . data ; num2 = num2 . next ; } Node newNode = new Node (sum % 10); carry = sum / 10 ; // If this is the first node, then make this node // as the head of the resultant linked list if (res == null) { res = newNode ; curr = newNode ; } else { // Append new node to resultant linked list // and move to the next node curr . next = newNode ; curr = curr . next ; } } return reverse (res); } static void printList (Node head) { Node curr = head ; while (curr != null) { System . out . print (curr . data); if (curr . next != null){ System . out . print (" -> "); } curr = curr . next ; } System . out . println (); } public static void main (String [] args) { Node num1 = new Node (1); num1 . next = new Node (2); num1 . next . next = new Node (3); Node num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); Node sum = addTwoLists (num1 , num2); printList (sum); } } Python class Node : def __init__ (self , val): self . data = val self . next = None # function to reverse the linked list def reverse (head): prev = None curr = head while curr is not None : nextNode = curr . next curr . next = prev prev = curr curr = nextNode return prev # function to trim leading zeros in linked list def trimLeadingZeros (head): while head and head . data == 0 : head = head . next return head # Function to add two numbers represented by linked list def addTwoLists (num1 , num2): res = None curr = None carry = 0 num1 = trimLeadingZeros (num1) num2 = trimLeadingZeros (num2) num1 = reverse (num1) num2 = reverse (num2) # Iterate till either num1 is not empty or num2 is # not empty or carry is greater than 0 while num1 is not None or num2 is not None or carry != 0 : sumValue = carry if num1 is not None : sumValue += num1 . data num1 = num1 . next if num2 is not None : sumValue += num2 . data num2 = num2 . next newNode = Node (sumValue % 10) carry = sumValue // 10 # If this is the first node, then make this node # as the head of the resultant linked list if res is None : res = newNode curr = newNode else : # Append new node to resultant linked list # and move to the next node curr . next = newNode curr = curr . next return reverse (res) def printList (head): curr = head while curr is not None : print (curr . data , end = "") if curr . next != None : print (" -> " , end = "") curr = curr . next print () if __name__ == "__main__" : num1 = Node (1) num1 . next = Node (2) num1 . next . next = Node (3) num2 = Node (9) num2 . next = Node (9) num2 . next . next = Node (9) sumList = addTwoLists (num1 , num2) printList (sumList) C# using System ; class Node { public int data ; public Node next ; public Node (int val) { data = val ; next = null ; } } class GfG { // Function to reverse the linked list static Node reverse (Node head) { Node prev = null ; Node curr = head ; Node next ; while (curr != null) { next = curr . next ; curr . next = prev ; prev = curr ; curr = next ; } return prev ; } // function to trim leading zeros in linked list static Node trimLeadingZeros (Node head) { while (head != null && head . data == 0) { head = head . next ; } return head ; } // Function to add two numbers represented by linked list static Node addTwoLists (Node num1 , Node num2) { Node res = null ; Node curr = null ; int carry = 0 ; num1 = trimLeadingZeros (num1); num2 = trimLeadingZeros (num2); num1 = reverse (

num1); num2 = reverse (num2); // Iterate till either num1 is not empty or num2 is // not empty or carry is greater than 0 while (num1 != null || num2 != null || carry != 0) { int sum = carry ; if (num1 != null) { sum += num1 . data ; num1 = num1 . next ; } if (num2 != null) { sum += num2 . data ; num2 = num2 . next ; } Node newNode = new Node (sum % 10); carry = sum / 10 ; // If this is the first node, then make this node // as the head of the resultant linked list if (res == null) { res = newNode ; curr = newNode ; } else { // Append new node to resultant linked list // and move to the next node curr . next = newNode ; curr = curr . next ; } } return reverse (res); } static void printList (Node head) { Node curr = head ; while (curr != null) { Console . Write (curr . data); if (curr . next != null){ Console . Write (" -> "); } curr = curr . next ; } Console . WriteLine (); } static void Main () { Node num1 = new Node (1); num1 . next = new Node (2); num1 . next . next = new Node (3); Node num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); Node sum = addTwoLists (num1 , num2); printList (sum); } } JavaScript class Node { constructor (val) { this . data = val ; this . next = null ; } } // Function to reverse the linked list function reverse (head) { let prev = null ; let curr = head ; let next ; while (curr !== null) { next = curr . next ; curr . next = prev ; prev = curr ; curr = next ; } return prev ; } // function to trim leading zeros in linked list function trimLeadingZeros (head) { while (head !== null && head . data === 0) { head = head . next ; } return head ; } // Function to add two numbers represented by linked list function addTwoLists (num1 , num2) { let res = null ; let curr = null ; let carry = 0 ; num1 = trimLeadingZeros (num1); num2 = trimLeadingZeros (num2); num1 = reverse (num1); num2 = reverse (num2); // Iterate till either num1 is not empty or num2 is // not empty or carry is greater than 0 while (num1 !== null || num2 !== null || carry !== 0) { let sum = carry ; if (num1 !== null) { sum += num1 . data ; num1 = num1 . next ; } if (num2 !== null) { sum += num2 . data ; num2 = num2 . next ; } let newNode = new Node (sum % 10); carry = Math . floor (sum / 10); // If this is the first node, then make this node // as the head of the resultant linked list if (res === null) { res = newNode ; curr = newNode ; } else { // Append new node to resultant linked list // and move to the next node curr . next = newNode ; curr = curr . next ; } } return reverse (res); } function printList (head) { let curr = head ; let result = " ; while (curr !== null) { result += curr . data ; if (curr . next !== null){ result += " -> " ; } curr = curr . next ; } console . log (result . trim ()); } // Driver Code let num1 = new Node (1); num1 . next = new Node (2); num1 . next . next = new Node (3); let num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); let sum = addTwoLists (num1 , num2); printList (sum); Output 1 -> 1 -> 2 -> 2 [Expected Approach] By storing sum on the longer list - O(m + n) Time and O(1) Space The idea is to first reverse both linked lists so that addition can be performed starting from the least significant digit. We then iterate through both lists simultaneously, adding corresponding digits along with any carry. For each sum, we create a new node and insert it at the front of the result list. After processing all nodes, if a carry remains, we add a new node for it. Finally, the resulting list represents the sum without any leading zeros. Illustrations: C++ #include <iostream> using namespace std ; class Node { public : int data ; Node * next ; Node (int val) { data = val ; next = nullptr ; } }; // Function to reverse a linked list Node * reverse (Node * head) { Node * prev = nullptr , * curr = head , * next ; while (curr != nullptr) { next = curr -> next ; curr -> next = prev ; prev = curr ; curr = next ; } return prev ; } Node * addTwoLists (Node * head1 , Node * head2) { // Reverse both lists to start from least significant digit head1 = reverse (head1); head2 = reverse (head2); Node * sum = NULL ; int carry = 0 ; // Traverse both lists until all digits and carry are processed while (head1 != NULL || head2 != NULL || carry != 0) { int newVal = carry ; if (head1) { newVal += head1 -> data ; head1 = head1 -> next ; } if (head2) { newVal += head2 -> data ; head2 = head2 -> next ; } carry = newVal / 10 ; newVal %= 10 ; // Insert the new digit at the front of the result list Node * newNode = new Node (newVal); newNode -> next = sum ; sum = newNode ; } // Remove leading zeros, if any while (sum != NULL && sum -> data == 0) { Node * temp = sum ; sum = sum -> next ; delete temp ; } // If result is empty, return single node with 0 if (sum == NULL) { return new Node (0); } return sum ; } void printList (Node * head) { Node * curr = head ; while (curr != nullptr) { cout << curr -> data ; if (curr -> next != NULL){ cout << " -> " ; } curr = curr -> next ; } cout << " \n " ; } int main () { Node * num1 = new Node (1); num1 -> next = new Node (2); num1 -> next -> next = new Node (3); Node * num2 = new Node (9); num2 -> next = new Node (9); num2 -> next -> next = new Node (9); Node * sum = addTwoLists (num1 , num2); printList (sum); return 0 ; } C #include <stdio.h> struct Node { int data ; struct Node * next ; }; struct Node * createNode (int val); // Function to create a new node struct Node * newNode (int data) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = data ; node -> next = NULL ; return node ; } // Function to reverse a linked list struct Node * reverse (struct Node * head) { struct Node * prev = NULL , * curr = head , * next ; while (curr != NULL) { next = curr -> next ; curr -> next = prev ; prev = curr ; curr = next ; } return prev ; } // Function to add two linked lists struct Node *

addTwoLists (struct Node * head1 , struct Node * head2) { // Reverse both lists to start // from least significant digit head1 = reverse (head1); head2 = reverse (head2); struct Node * sum = NULL ; int carry = 0 ; // Traverse both lists until all // digits and carry are processed while (head1 != NULL || head2 != NULL || carry != 0) { int newVal = carry ; if (head1) { newVal += head1 -> data ; head1 = head1 -> next ; } if (head2) { newVal += head2 -> data ; head2 = head2 -> next ; } carry = newVal / 10 ; newVal %= 10 ; // Insert the new digit at the front of the result list struct Node * node = newNode (newVal); node -> next = sum ; sum = node ; } // Remove leading zeros, if any while (sum != NULL && sum -> data == 0) { struct Node * temp = sum ; sum = sum -> next ; free (temp); } // If result is empty, return single node with 0 if (sum == NULL) { return newNode (0); } return sum ; } void printList (struct Node * head) { struct Node * curr = head ; while (curr != NULL) { printf ("%d" , curr -> data); if (curr -> next != NULL) printf (" -> "); curr = curr -> next ; } printf (" \n "); } struct Node * createNode (int val) { struct Node * node = (struct Node *) malloc (sizeof (struct Node)); node -> data = val ; node -> next = NULL ; return node ; } int main () { struct Node * num1 = createNode (1); num1 -> next = createNode (2); num1 -> next -> next = createNode (3); struct Node * num2 = createNode (9); num2 -> next = createNode (9); num2 -> next -> next = createNode (9); struct Node * sum = addTwoLists (num1 , num2); printList (sum); return 0 ; } Java class Node { int data ; Node next ; Node (int val) { data = val ; next = null ; } } class GfG { // Function to reverse the linked list static Node reverse (Node head) { Node prev = null , curr = head , next = null ; while (curr != null) { next = curr . next ; curr . next = prev ; prev = curr ; curr = next ; } return prev ; } static Node addTwoLists (Node head1 , Node head2) { // Reverse both lists to start // from least significant digit head1 = reverse (head1); head2 = reverse (head2); Node sum = null ; int carry = 0 ; // Traverse until both lists // and carry are processed while (head1 != null || head2 != null || carry > 0) { int newVal = carry ; if (head1 != null) { newVal += head1 . data ; head1 = head1 . next ; } if (head2 != null) { newVal += head2 . data ; head2 = head2 . next ; } carry = newVal / 10 ; newVal %= 10 ; // Create new node and insert // at front of result list Node newNode = new Node (newVal); newNode . next = sum ; sum = newNode ; } // Remove leading zeros if present while (sum != null && sum . data == 0) { sum = sum . next ; } return (sum == null) ? new Node (0) : sum ; } static void printList (Node head) { Node curr = head ; while (curr != null) { System . out . print (curr . data); if (curr . next != null){ System . out . print (" -> "); } curr = curr . next ; } System . out . println (); } public static void main (String [] args) { Node num1 = new Node (1); num1 . next = new Node (2); num1 . next . next = new Node (3); Node num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); Node sum = addTwoLists (num1 , num2); printList (sum); } } Python class Node : def __init__ (self , val): self . data = val self . next = None # Function to reverse the linked list def reverse (head): prev = None current = head while current is not None : next = current . next current . next = prev prev = current current = next return prev def addTwoLists (head1 , head2): # Reverse both lists to simplify addition head1 = reverse (head1) head2 = reverse (head2) sumList = None carry = 0 # Loop until both lists and carry are exhausted while head1 is not None or head2 is not None or carry > 0 : newVal = carry if head1 is not None : newVal += head1 . data head1 = head1 . next if head2 is not None : newVal += head2 . data head2 = head2 . next carry = newVal // 10 newVal = newVal % 10 # Create a new node and link it at the front newNode = Node (newVal) newNode . next = sumList sumList = newNode # Return the final sum list return sumList def printList (head): curr = head while curr is not None : print (curr . data , end = "") if curr . next is not None : print (" -> " , end = "") curr = curr . next print () if __name__ == "__main__" : num1 = Node (1) num1 . next = Node (2) num1 . next . next = Node (3) num2 = Node (9) num2 . next = Node (9) num2 . next . next = Node (9) sumList = addTwoLists (num1 , num2) printList (sumList) C# using System ; class Node { public int data ; public Node next ; public Node (int val) { data = val ; next = null ; } } class GfG { // Function to reverse a linked list static Node reverse (Node head) { Node prev = null ; Node current = head ; Node next ; while (current != null) { next = current . next ; current . next = prev ; prev = current ; current = next ; } return prev ; } static Node addTwoLists (Node head1 , Node head2) { // Reverse both lists to simplify addition head1 = reverse (head1); head2 = reverse (head2); Node sum = null ; int carry = 0 ; // Loop while nodes remain or carry is non-zero while (head1 != null || head2 != null || carry != 0) { int newVal = carry ; if (head1 != null) { newVal += head1 . data ; head1 = head1 . next ; } if (head2 != null) { newVal += head2 . data ; head2 = head2 . next ; } carry = newVal / 10 ; newVal = newVal % 10 ; Node newNode = new Node (newVal); newNode . next = sum ; sum = newNode ; } // Remove leading zeros while (sum != null && sum . data == 0) { sum = sum . next ; } return (sum != null) ? sum : new Node (0); } static void printList (Node head) { Node curr = head ; while (curr != null) { Console . Write (curr . data); if (curr . next != null){ Console . Write (" -> "); } curr = curr . next ; } Console . WriteLine (); } static void Main () { Node num1 = new Node (1); num1 . next = new Node (2

); num1 . next . next = new Node (3); Node num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); Node sum = addTwoLists (num1 , num2); printList (sum); } } JavaScript class Node { constructor (val) { this . data = val ; this . next = null ; } } // function to reverse a linked list function reverse (head) { let prev = null ; let current = head ; let next ; while (current !== null) { next = current . next ; current . next = prev ; prev = current ; current = next ; } return prev ; } function addTwoLists (head1 , head2) { // reverse both lists to start // from least significant digit head1 = reverse (head1); head2 = reverse (head2); let sum = null ; let carry = 0 ; // loop until both lists and // carry are exhausted while (head1 !== null || head2 !== null || carry !== 0) { let newVal = carry ; if (head1) { newVal += head1 . data ; head1 = head1 . next ; } if (head2) { newVal += head2 . data ; head2 = head2 . next ; } carry = Math . floor (newVal / 10); newVal = newVal % 10 ; let newNode = new Node (newVal); newNode . next = sum ; sum = newNode ; } // remove leading zeros while (sum !== null && sum . data === 0) { sum = sum . next ; } return sum === null ? new Node (0) : sum ; } function printList (head) { let curr = head ; let result = "" ; while (curr !== null) { result += curr . data ; if (curr . next !== null){ result += " -> " ; } curr = curr . next ; } console . log (result . trim ()); } // Driver Code let num1 = new Node (1); num1 . next = new Node (2); num1 . next . next = new Node (3); let num2 = new Node (9); num2 . next = new Node (9); num2 . next . next = new Node (9); let sum = addTwoLists (num1 , num2); printList (sum); Output 1 -> 1 -> 2 -> 2 [Other Approach] Using Recursion - O(m + n) Time and O(max(m, n)) Space The idea is to use recursion to compute the sum . Recursively move to the end of the lists, calculate the sum of the last nodes (including any carry from previous additions), while backtracking add up the sums together. For a more detailed solution and code, check this article Add two numbers represented as Linked List using Recursion . Comment Article Tags: Article Tags: Linked List DSA Microsoft Amazon Flipkart Qualcomm Snapdeal Accolite MakeMyTrip + 5 More