**Assignment**

**Linked Lists in Java**

**Q1. Given a linked list and a key ‘X‘ in, the task is to check if X is present in the linked list or not.**

**Examples:**

**Input: 14->21->11->30->10, X = 14**

**Output: Yes**

**Explanation: 14 is present in the linked list.**

**Input: 6->21->17->30->10->8, X = 13**

**Output: No**

**Solution:**

Search an element in a Linked List (Iterative Approach):

Follow the below steps to solve the problem:

* Initialize a node pointer, current = head.
* Do following while current is not NULL
* If the current value (i.e., current->key) is equal to the key being searched return true.
* Otherwise, move to the next node (current = current->next).
* If the key is not found, return false

**Time Complexity:** O(N), Where N is the number of nodes in the LinkedList  
**Auxiliary Space:** O(1

**package** LinkedList.AssignmentLinkedList;

**import** java.util.LinkedList;

**public** **class** AssignmentProblem1

{

**class** Node

{

**int** data;

Node next;

Node(**int** d)

{

data = d;

next = **null**;

}

}

Node head;

**public** **void** push(**int** new\_data)

{

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

new\_node.next = head;

head = new\_node;

}

**public** **boolean** search(Node head, **int** x)

{

Node current = head;

**while** (current != **null**)

{

**if** (current.data == x)

**return** **true**;

current = current.next;

}

**return** **false**;

}

**public** **static** **void** main(String args[])

{

AssignmentProblem1 llist = **new** AssignmentProblem1();

**int** x = 14;

llist.push(14);

llist.push(21);

llist.push(11);

llist.push(30);

llist.push(10);

**if** (llist.search(llist.head, x))

System.***out***.println("Yes");

**else**

System.***out***.println("No");

}

}

**Q2. Insert a node at the given position in a linked list. We are given a pointer to a node, and the new node is inserted after the given node.**

**Input : LL = 1 -> 2 -> 4 -> 5 -> 6 pointer = 2     value =  3.**

**Output :1 -> 2 -> 3 -> 4 -> 5 -> 6**

**Solution:**

**Approach:**

To insert a node after a given node in a Linked List, we need to:

* Check if the given node exists or not.
  + If it do not exists,
    - Terminate the process.
  + If the given node exists,
    - Make the element to be inserted as a new node
    - Change the next pointer of given node to the new node
    - Now shift the original next pointer of given node to the next pointer of new node

Follow the below steps for inserting a node after a given node:

* Firstly, check if the given previous node is NULL or not.
* Then, allocate a new node (say **temp**) and
* Assign the data to **temp**.
* And then make the next of **temp** as the next of the previous node.
* Finally, move the next of the previous node to point to **temp**.

**Implementaton**

**package** LinkedList.AssignmentLinkedList;

//Time Complexity: O(1)  
// Space Complexity: O(1)

**import** java.util.LinkedList;

**import** java.io.\*;

**class** Node {

**int** data;

Node next;

}

**public** **class** AssignmentProblem2 {

Node head;

**void** insertAfter(Node prev\_node, **int** new\_data) {

**if** (prev\_node == **null**) {

System.***out***.println

("The given previous node cannot be NULL");

**return**;

}

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

new\_node.data = new\_data;

new\_node.next = prev\_node.next;

prev\_node.next = new\_node;

}

**void** push(**int** new\_data) {

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

new\_node.data = new\_data;

new\_node.next = head;

head = new\_node;

}

**void** printList() {

Node node = head;

**while** (node != **null**) {

System.***out***.print(" " + node.data);

node = node.next;

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

AssignmentProblem2 llist = **new** AssignmentProblem2();

llist.push(6);

llist.push(5);

llist.push(4);

llist.push(2);

System.***out***.print("Created Linked list is: ");

llist.printList();

llist.insertAfter(llist.head, 3);

System.***out***.print("After inserting 1 after 2: ");

llist.printList();

}

}

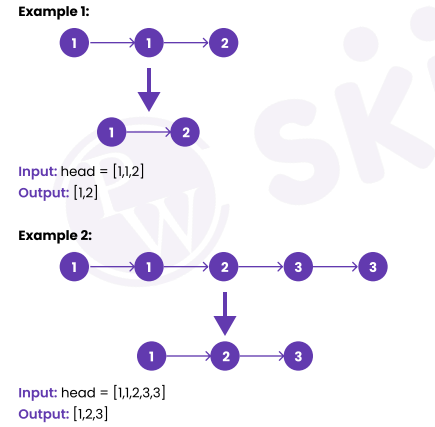
/\* Output:

Created Linked list is: 2 4 5 6

After inserting 1 after 2: 2 3 4 5 6

\*/

**Q3. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.**

****

**Solution:**

**Approach:**

Create a pointer that will point towards the first occurrence of every element and another pointer temp which will iterate to every element and when the value of the previous pointer is not equal to the temp pointer, we will set the pointer of the previous pointer to the first occurrence of another node.

**Implementation**

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

**// Space Complexity:** O(1)

**package** LinkedList.AssignmentLinkedList;

**import** java.io.\*;

**class** AssignmentProblem3

{

Node head;

**class** Node

{

**int** data;

Node next;

Node(**int** d)

{

data = d;

next = **null**;

}

}

**void** removeDuplicates()

{

Node temp = head, prev = head;

**while** (temp != **null**)

{

**if** (temp.data != prev.data)

{

prev.next = temp;

prev = temp;

}

temp = temp.next;

}

**if** (prev != temp)

prev.next = **null**;

}

**public** **void** push(**int** new\_data)

{

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

new\_node.next = head;

head = new\_node;

}

**void** printList()

{

Node temp = head;

**while** (temp != **null**) {

System.***out***.print(temp.data + " ");

temp = temp.next;

}

System.***out***.println();

}

**public** **static** **void** main(String args[])

{

AssignmentProblem3 llist = **new** AssignmentProblem3();

llist.push(3);

llist.push(3);

llist.push(2);

llist.push(1);

llist.push(1);

System.***out***.print("List before removal of duplicates :");

llist.printList();

llist.removeDuplicates();

System.***out***.print("List after removal of elements: ");

llist.printList();

}

}

/\*

Output:

List before removal of duplicates :1 1 2 3 3

List after removal of elements: 1 2 3

\*/

Q4. Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example 1:

Input: head = [1, 2, 2, 1]

Output: true

Example 2:

Input: head = [1, 2]

Output: false

**Solution:**

Following are the steps to this approach.

* Iterate through the given list to store it in an array.
* Iterate through the array.
* For each index in range of n/2 where n is the size of the array
* Check if the number in it is the same as the number in the n-index-1 of the array.

**Implementation**

**package** LinkedList.AssignmentLinkedList;

**import** java.util.ArrayList;

**public** **class** AssignmentProblem4

{

**static** **class** Node

{

**int** num;

Node next;

Node(**int** val)

{

num = val;

next = **null**;

}

}

**static** Node insertNode(Node head,**int** val)

{

Node newNode = **new** Node(val);

**if**(head == **null**)

{

head = newNode;

**return** head;

}

Node temp = head;

**while**(temp.next != **null**) temp = temp.next;

temp.next = newNode;

**return** head;

}

**static** **boolean** isPalindrome(Node head)

{

ArrayList<Integer> arr=**new** ArrayList<>();

**while**(head != **null**)

{

arr.add(head.num);

head = head.next;

}

**for**(**int** i=0;i<arr.size()/2;i++)

**if**(arr.get(i) != arr.get(arr.size()-i-1)) **return** **false**;

**return** **true**;

}

**public** **static** **void** main(String args[])

{

Node head = **null**;

head=*insertNode*(head,1);

head=*insertNode*(head,2);

head=*insertNode*(head,3);

head=*insertNode*(head,2);

head=*insertNode*(head,1);

**if**(*isPalindrome*(head)==**true**)

System.***out***.println("True");

**else**

System.***out***.println("False");

}

}

/\*

Output:

List before removal of duplicates :1 1 2 3 3

List after removal of elements: 1 2 3

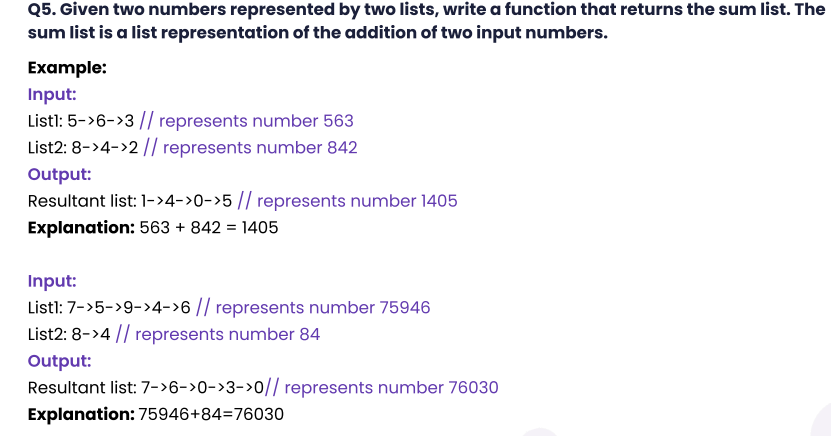
\*/

Time Complexity: O(N)

Reason: Iterating through the list to store elements in the array.

Space Complexity: O(N)

Reason: Using an array to store list elements for further computations.

****

**Solution:**

Approah :

Traverse both lists to the end and add preceding zeros in the list with lesser digits.

Then call a recursive function on the start nodes of both lists which calls itself for the next nodes of both lists till it gets to the end.

This function creates a node for the sum of the current digits and returns the carry.

Traverse the two linked lists in order to add preceding zeros in case a list is having lesser digits than the other one.

Start from the head node of both lists and call a recursive function for the next nodes.

Continue it till the end of the lists.

Creates a node for current digits sum and returns the carry.

**Implementation**

**package** LinkedList.AssignmentLinkedList;

**public** **class** AssignmentProblem5

{

**static** Node *head1*, *head2*;

**static** **class** Node

{

**int** data;

Node next;

Node(**int** d)

{

data = d;

next = **null**;

}

}

**void** addTwoLists(Node first, Node second)

{

Node start1 = **new** Node(0);

start1.next = first;

Node start2 = **new** Node(0);

start2.next = second;

addPrecedingZeros(start1, start2);

Node result = **new** Node(0);

**if** (sumTwoNodes(start1.next, start2.next, result) == 1)

{

Node node = **new** Node(1);

node.next = result.next;

result.next = node;

}

printList(result.next);

}

**private** **int** sumTwoNodes(Node first, Node second, Node result)

{

**if** (first == **null**)

{

**return** 0;

}

**int** number = first.data + second.data + sumTwoNodes(first.next, second.next, result);

Node node = **new** Node(number % 10);

node.next = result.next;

result.next = node;

**return** number / 10;

}

**private** **void** addPrecedingZeros(Node start1, Node start2)

{

Node next1 = start1.next;

Node next2 = start2.next;

**while** (next1 != **null** && next2 != **null**)

{

next1 = next1.next;

next2 = next2.next;

}

**if** (next1 == **null** && next2 != **null**)

{

**while** (next2 != **null**)

{

Node node = **new** Node(0);

node.next = start1.next;

start1.next = node;

next2 = next2.next;

}

} **else** **if** (next2 == **null** && next1 != **null**)

{

**while** (next1 != **null**)

{

Node node = **new** Node(0);

node.next = start2.next;

start2.next = node;

next1 = next1.next;

}

}

}

**void** printList(Node head)

{

**while** (head != **null**)

{

System.***out***.print(head.data + " ");

head = head.next;

}

System.***out***.println("");

}

**public** **static** **void** main(String[] args)

{

AssignmentProblem5 list = **new** AssignmentProblem5();

// creating first list

list.*head1* = **new** Node(5);

list.*head1*.next = **new** Node(6);

list.*head1*.next.next = **new** Node(3);

System.***out***.print("First List is : ");

list.printList(*head1*);

// creating second list

list.*head2* = **new** Node(8);

list.*head2*.next = **new** Node(4);

list.*head2*.next.next = **new** Node(2);

System.***out***.print("Second List is : ");

list.printList(*head2*);

System.***out***.print("Resultant List is : ");

// add the two lists and see the result

list.addTwoLists(*head1*, *head2*);

}

}

/\*

Output:

First List is : 5 6 3

Second List is : 8 4 2

Resultant List is : 1 4 0 5

\*/