Linked Lists in Java

Assignment Solutions

PW SKILLS

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 :

Code : ASS_Code1.java

Output :

```
Yes
```

Approach :

- We have simply traversed the linked list and checked for every node whether its data is equal to the target.
- If found we returned true else if the iterator reached the end of the linked list we returned false.

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 :

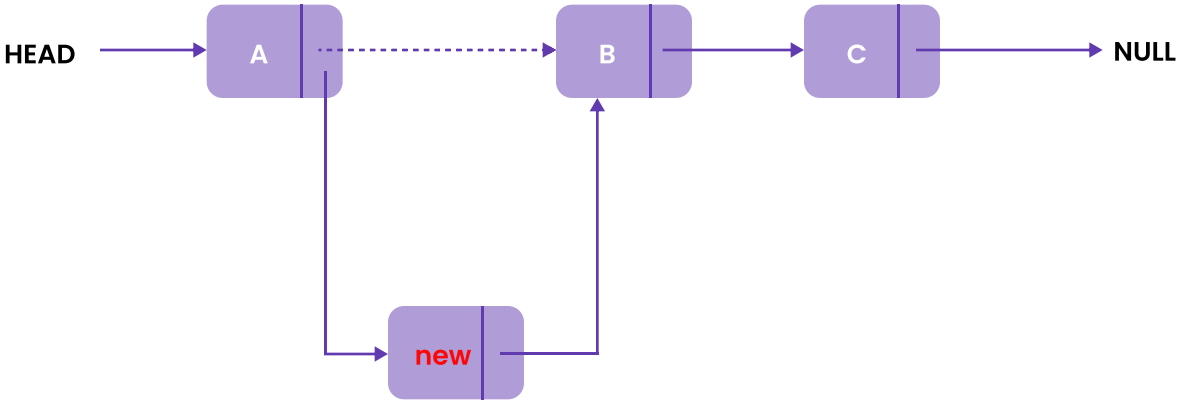
Code : ASS_Code2.java

Output :

```
The list contains: 3 6 5
The list contains: 3 8 6 5
The list contains: 3 8 9 6 5
```

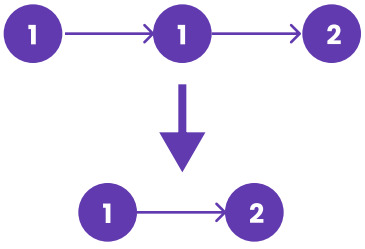
Approach :

- First, a new node with a given element is created. If the insert position is 1, then the new node is made to head.
- Otherwise, traverse to the node that is previous to the insert position and check if it is null or not. In case of null, the specified position does not exist.
- In other case, assign the next of the new node as the next of the previous node and the next of the previous node as new node.
- The below figure describes the process, if the insert node is other than the head node.



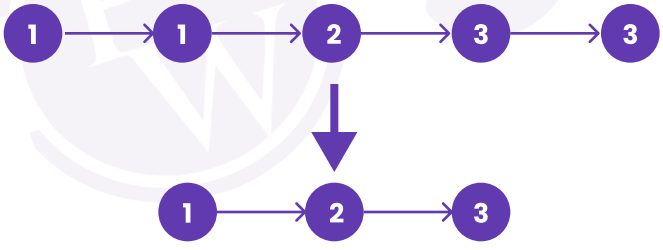
Q3. Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

Example 1:



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

Example 2:



Input: head = [1,1,2,3,3]
Output: [1,2,3]

Solution :

Code : ASS_Code3.java

Output :

```
The list contains: 3 3 5 6 6 7
The list contains: 3 5 6 7
```

Approach :

- Traverse all element through a while loop if curr node and the next node of curr node are present.
- If the value of curr is equal to the value of prev.

- It means the value is present in the linked list.
- So we can skip the 'curr' pointer and prev.next can point to curr.next. By this we ensure that prev now doesn't points to curr anymore. The node curr is not pointed anymore, so it would be garbage collected automatically by JVM.
- Hence we do not need to include curr again in the linked list, so we increment the value of curr otherwise, we increment the curr pointer.

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 :

Code : ASS_Code4.java

Output :

```
Is Palindrome
```

Approach :

- The idea is to first reverse the second half part of the linked list and then check whether the list is palindrome or not.
- Get the middle of the linked list and reverse the second half of the linked list.
- Check if the first half and second half are identical.
- Construct the original linked list by reversing the second half again and attaching it back to the first half

Q5. Given two numbers represented by two lists, write a function that returns the sum list. The sum list is a list representation of the addition of two input numbers.

Example:
Input:
List1: 5->6->3 // represents number 563
List2: 8->4->2 // represents number 842
Output:
Resultant list: 1->4->0->5 // represents number 1405
Explanation: 563 + 842 = 1405

Input:
List1: 7->5->9->4->6 // represents number 75946

List2: 8->4 // represents number 84

Output:

Resultant list: 7->6->0->3->0// represents number 76030

Explanation: 75946+84=76030

Solution :

Code : ASS_Code5.java

Output :

```
First List is 7 5 9 4 6
Second List is 8 4
Resultant List is 7 6 0 3 0
```

Approach :

- 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.