Assignment_12_LinkedList_Solution

Question 1

Given a singly linked list, delete **middle** of the linked list. For example, if given linked list is 1->2->**3**->4->5 then linked list should be modified to 1->2->4->5.If there are **even** nodes, then there would be **two middle** nodes, we need to delete the second middle element. For example, if given linked list is 1->2->3->4->5->6 then it should be modified to 1->2->3->5->6.If the input linked list is NULL or has 1 node, then it should return NULL

Solution:

```
public ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }
        ListNode slowPointer = head;
        ListNode fastPointer = head;
        ListNode previous = null;
        while (fastPointer != null && fastPointer.next != null) {
            previous = slowPointer;
            slowPointer = slowPointer.next;
            fastPointer = fastPointer.next.next;
        }
        previous.next = slowPointer.next;
        return head;
    }
```

Question 2

Given a linked list of **N** nodes. The task is to check if the linked list has a loop. Linked list can contain self loop.

Solution:

```
public boolean hasCycle(ListNode head) {

        ListNode fast = head;
        ListNode slow = head;

        boolean cycle = false;

        //detect cycle
        while(fast != null && fast.next != null){

            if(fast.val == slow.val){
                    cycle = true;
                    break;
                // break;
            }

            slow = slow.next;
            fast = fast.next.next;
        }

        if(slow == null || fast == null){
            return false;
        }
```

```
        return true;



    }
```

Question 3

Given a linked list consisting of **L** nodes and given a number **N**. The task is to find the **N**th node from the end of the linked list

Solution:

```
int printNthFromLast(int N)
    {
        int len = 0;
        Node temp = head;


        while (temp != null) {
            temp = temp.next;
            len++;
        }



        if (len < N)
            return;
```

```
        temp = head;


        for (int i = 1; i < len - N + 1; i++)
            temp = temp.next;


        return temp.data
    }
```

Question 4

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

Solution:

```
public boolean isPalindrome(ListNode head) {
        if(head == null || head.next == null){
            return true;
        }
        ListNode slow = head;
        ListNode fast = head;


        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
```

```
ListNode curr = slow;
ListNode prev = null;
ListNode next;

while(curr != null){
    next = curr.next;
    curr.next = prev;
    prev = curr;
    curr = next;
}

ListNode head1 = head;
ListNode head2 = prev;

while(head2 != null && head1 != null){
    if(head1.val != head2.val){
        return false;
    }

    head1 = head1.next;
    head2 = head2.next;
}
```

```
        return true;



    }
```

Question 5

Given a linked list of **N** nodes such that it may contain a loop.

A loop here means that the last node of the link list is connected to the node at position X(1-based index). If the link list does not have any loop, X=0.

Remove the loop from the linked list, if it is present, i.e. unlink the last node which is forming the loop.

Solution:

```
 int detectAndRemoveLoop(Node node)
    {
        Node slow = node, fast = node;
        while (slow != null && fast != null
               && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            // If slow and fast meet at same point then loop
            // is present
```

```
        if (slow == fast) {
            removeLoop(slow, node);
            return 1;
        }
    }
    return 0;
}


void removeLoop(Node loop, Node head)
{
    Node ptr1 = loop;
    Node ptr2 = loop;

    int k = 1, i;
    Node prevNode = ptr1;
    while (ptr1.next != ptr2) {

        prevNode = ptr1;
        ptr1 = ptr1.next;
        k++;
    }
    prevNode.next = null;
```

```
}
```

Question 6

Given a linked list and two integers M and N. Traverse the linked list such that you retain M nodes then delete next N nodes, continue the same till end of the linked list.

Solution:

```
static void skipMdeleteN( Node head, int M, int N)
{
    Node curr = head, t;
    int count;

    while (curr!=null)
    {

        for (count = 1; count < M && curr != null; count++)
            curr = curr.next;


        if (curr == null)
            return;


        t = curr.next;
```

```
        for (count = 1; count <= N && t != null; count++)
        {
            Node temp = t;
            t = t.next;
        }



        curr.next = t;



        curr = t;
    }
}
```

Question 7

Given two linked lists, insert nodes of second list into first list at alternate positions of first list. For example, if first list is 5->7->17->13->11 and second is 12->10->2->4->6, the first list should become 5->12->7->10->17->2->13->4->11->6 and second list should become empty. The nodes of second list should only be inserted when there are positions available. For example, if the first list is 1->2->3 and second list is 4->5->6->7->8, then first list should become 1->4->2->5->3->6 and second list to 7->8.

Use of extra space is not allowed (Not allowed to create additional nodes), i.e., insertion must be done in-place. Expected time complexity is O(n) where n is number of nodes in first list.

Solution:

```
void merge(LinkedList q)
    {
        Node p_curr = head, q_curr = q.head;
        Node p_next, q_next;



        while (p_curr != null && q_curr != null) {



            p_next = p_curr.next;
            q_next = q_curr.next;



            q_curr.next = p_next; // change next pointer of q_curr
            p_curr.next = q_curr; // change next pointer of p_curr



            p_curr = p_next;
            q_curr = q_next;
        }
        q.head = q_curr;
    }
```

Question 8

Given a singly linked list, find if the linked list is circular 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.

Solution:

```
 static boolean isCircular(Node head)
   {


      if (head == null)
         return true;


      // Next of head
      Node node = head.next;



      while (node != null && node != head)
         node = node.next;



      return (node == head);
   }
```