Assignment 13 LinkList Solution

Question 1

Given two linked list of the same size, the task is to create a new linked list using those linked lists. The condition is that the greater node among both linked list will be added to the new linked list.

Solution:

```
static Node insert(Node root, int item)

{

    Node ptr, temp;

    temp = new Node();

    temp.data = item;

    temp.next = null;


    if (root == null)

        root = temp;

    else {

        ptr = root;

        while (ptr.next != null)

            ptr = ptr.next;


        ptr.next = temp;

    }
```

```
    return root;
}
static Node newList(Node root1, Node root2)
{
    Node ptr1 = root1, ptr2 = root2, ptr;
    Node root = null, temp;

    while (ptr1 != null) {
        temp = new Node();
        temp.next = null;

        // Compare for greater node
        if (ptr1.data < ptr2.data)
            temp.data = ptr2.data;
        else
            temp.data = ptr1.data;

        if (root == null)
            root = temp;
        else {
            ptr = root;
            while (ptr.next != null)
                ptr = ptr.next;
```

```
            ptr.next = temp;
        }
        ptr1 = ptr1.next;
        ptr2 = ptr2.next;
    }
    return root;
}
```

Question 2

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.

Solution:

```
 public ListNode deleteDuplicates(ListNode head) {
       if(head==null||head.next==null)
       return head;
        ListNode left = head;
        ListNode right = head.next;
        int node1=left.val,node2=right.val;
        while(right!=null){
            node1 = left.val;
```

```
            node2= right.val;
            if(node1==node2){
                left.next = right.next;
                right=right.next;
            }
            if(node1!=node2){
                left = left.next;
                right = right.next;
            }

        }
        return head;
    }
```

Question 3

Given a linked list of size **N**. The task is to reverse every **k** nodes (where k is an input to the function) in the linked list. If the number of nodes is not a multiple of *k* then left-out nodes, in the end, should be considered as a group and must be reversed (See Example 2 for clarification).

Solution:

```
public:
    ListNode* reverseKGroup(ListNode* head, int k) {
        ListNode* dummy = new ListNode(0);
        dummy->next = head;
        ListNode* prevGroupTail = dummy;
```

```
        while (head) {
            ListNode* groupStart = head;
            ListNode* groupEnd = getGroupEnd(head, k);

            if (!groupEnd) {
                break; // Remaining nodes are less than k, so no need to
reverse
            }

            ListNode* nextGroupStart = groupEnd->next;
            groupEnd->next = nullptr; // Separate the group to be reversed

            // Reverse the group
            prevGroupTail->next = reverseList(groupStart);
            groupStart->next = nextGroupStart;

            prevGroupTail = groupStart;
            head = nextGroupStart;
        }

        ListNode* newHead = dummy->next;
        delete dummy;
```

```
        return newHead;
    }

private:
    ListNode* getGroupEnd(ListNode* head, int k) {
        while (head && k > 1) {
            head = head->next;
            k--;
        }
        return head;
    }

    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* curr = head;

        while (curr) {
            ListNode* next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
```

```
    return prev;
}
```

Question 4

Given a linked list, write a function to reverse every alternate k nodes (where k is an input to the function) in an efficient way. Give the complexity of your algorithm.

Solution:

Question 5

Given a linked list and a key to be deleted. Delete last occurrence of key from linked. The list may have duplicates.

Solution:

```
static Node deleteLast(Node head,int x)
{
    Node temp = head;
    Node ptr = null;

    while (temp != null)
    {

        // If found key, update
        if (temp.data == x)
            ptr = temp;
```

```
    temp = temp.next;
}

// If the last occurrence is the last node
if (ptr != null && ptr.next == null)
{
    temp = head;

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

// If it is not the last node
if (ptr != null && ptr.next != null)
{
    ptr.data = ptr.next.data;
    temp = ptr.next;
    ptr.next = ptr.next.next;
}
return head;
```

}

Question 6

Given two sorted linked lists consisting of **N** and **M** nodes respectively. The task is to merge both of the lists (in place) and return the head of the merged list.

Solution:

```
static Node deleteLast(Node head,int x)

{

    Node temp = head;

    Node ptr = null;


    while (temp != null)

    {


        // If found key, update

        if (temp.data == x)

            ptr = temp;


        temp = temp.next;

    }


    // If the last occurrence is the last node

    if (ptr != null && ptr.next == null)

    {
```

```
        temp = head;


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


    // If it is not the last node
    if (ptr != null && ptr.next != null)
    {
        ptr.data = ptr.next.data;
        temp = ptr.next;
        ptr.next = ptr.next.next;
    }
    return head;
}
```

Question 7

Given a **Doubly Linked List**, the task is to reverse the given Doubly Linked List.

Solution:

Node reverse(Node head)

```
{
    Node prev = null;


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



    return prev;
}
```

Question 8

Given a doubly linked list and a position. The task is to delete a node from given position in a doubly linked list.

Solution:

```
 static Node deleteNode(Node del)
    {
        // base case
        if (head == null || del == null)
```

```
        return null;

    // If node to be deleted is head node
    if (head == del)
        head = del.next;
    if (del.next != null)
        del.next.prev = del.prev;

    if (del.prev != null)
        del.prev.next = del.next;

    del = null;

    return head;
}

static void deleteNodeAtGivenPos(int n)
{

    if (head == null || n <= 0)
        return;

    Node current = head;
```

```
    int i;



    for (i = 1; current != null && i < n; i++)
    {
        current = current.next;
    }

    if (current == null)
        return;



    deleteNode(current);
}
```