Tutorial 2 – Linked List

**Question 1.** Assume the structure of a Linked List node is as follows.

```c
struct node {
    int data;
    node * next;
}
```

Suppose that we have a linked lists as shown in the following figure:

Draw the linked list in which case:

a) Insert a node (value of data: 9) at the beginning of linked list.
b) Insert a node (value of data: 15) at the end of linked list.
c) Insert a node (value of data: 10) at the pTemp.
d) Delete the node which have value of data 3.
e) Delete the node which have value of data 17.
f) Delete the node which pTemp pointed.

What is the output of the following code?

```
g) void fun1(node* head) {
    if (head == NULL)
        return;
    fun1(head->next);
    printf("%d  ", head->data);
}
h) void fun2(node* head) {
    if (head == NULL)
        return;
    printf("%d  ", head->data);
    if (head->next != NULL)
        fun2(head->next->next);
    printf("%d  ", head->data);
}
```

**Answer:**

```
a) 9  3  6  13  5  17  NULL
```

503001 – Data Structure & Algorithms – Tutorial 2

CuuDuongThanCong.com

https://fb.com/tailieudientucntt
Question 2. Imagine we have the method to delete an element from a list as shown in the figure. Which of the following code can be used in searching the target element (i.e. pred and tmp pointers)? Explain.

a. 1. pred=null; tmp = head;
   2. loop (tmp is not null and tmp->data is not the target)
      1. pred = pred->next;
      2. tmp = tmp->next;
      3. end loop

b. 1. pred=null; tmp = head;
   2. loop (tmp is not null and tmp->data is not the target)
      1. pred = tmp;
      2. tmp = tmp->next;
      3. end loop

Answer:
The code segment in (a) cannot be selected because there is an error when the statement pred = pred->next is executed (with pred = null). In the code segment b, pred is used to store the current value of tmp, which is correct.
Question 3. Write a member function which takes a list sorted in increasing order and deletes any duplicate nodes from the list.

Example: Input: 1->1->5->7->7->9->10->NULL
Output: 1->5->7->9->10->NULL

Answer:
```c
void removeDuplicates(node* head) {
    node* current = head;
    node* nextNext;
    if (current == NULL)
        return;
    while(current->next!=NULL)
        if (current->data == current->next->data) {
            nextNext = current->next;
            current->next = current->next->next;
            free(nextNext);
        } else {
            current = current->next;
        }
}
```

Question 4. Write a member function to remove all the nodes which have a greater value on right side.

Example: Input: 12->15->10->11->5->6->2->3->NULL
Output: 15->11->6->3->NULL

Answer: Use two loops. In the outer loop, pick nodes of the linked list one by one. In the inner loop, check if there exist a node whose value is greater than the picked node. If there exists a node whose value is greater, then delete the picked node.

```c
void delLesserNodes(node *head) {
    node *current = head;
    /* Initialize max */
    node *maxnode = head;
    node *temp;
    while (current != NULL && current->next != NULL) {
        /* If current is smaller than max, then delete current */
        if (current->next->data < maxnode->data) {
            temp = current->next;
            current->next = temp->next;
            delete temp;
        } else {
            /* If current is greater than max, then update max and move current */
            current = current->next;
            maxnode = current;
        }
    }
}
```
**Question 5.** Write a member function that takes two lists and insert nodes of second list into first list at alternate positions of first list.

*Note: Use of extra space is not allowed (Not allowed to create additional nodes).*

**Example:**

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1: 1-&gt;2-&gt;5-&gt;NULL</td>
<td>L1: 1-&gt;4-&gt;2-&gt;10-&gt;5-&gt;13-&gt;NULL</td>
</tr>
<tr>
<td>L2: 4-&gt;10-&gt;13-&gt;NULL</td>
<td>L2: NULL</td>
</tr>
<tr>
<td>L1: 1-&gt;2-&gt;5-&gt;NULL</td>
<td>L1: 1-&gt;4-&gt;2-&gt;10-&gt;5-&gt;13-&gt;NULL</td>
</tr>
<tr>
<td>L2: 4-&gt;10-&gt;13-&gt;20-&gt;NULL</td>
<td>L2: 20-&gt;NULL</td>
</tr>
<tr>
<td>L1: 1-&gt;2-&gt;5-&gt;7-&gt;NULL</td>
<td>L1: 1-&gt;4-&gt;2-&gt;10-&gt;5-&gt;7-&gt;NULL</td>
</tr>
<tr>
<td>L2: 4-&gt;10-&gt;NULL</td>
<td>L2: NULL</td>
</tr>
</tbody>
</table>

**Answer:**

```c
void merge(struct node *p, struct node **q) {
    node *p_curr = p, *q_curr = *q;
    node *p_next, *q_next;

    // While there are available positions in p
    while (p_curr != NULL && q_curr != NULL) {
        // Save next pointers
        p_next = p_curr->next;
        q_next = q_curr->next;

        // Make q_curr as next of p_curr
        q_curr->next = p_next; // Change next pointer of q_curr
        p_curr->next = q_curr; // Change next pointer of p_curr

        // Update current pointers for next iteration
        p_curr = p_next;
        q_curr = q_next;
    }

    *q = q_curr; // Update head pointer of second list
}
```