Flatten a multi-level linked list | Set 2 (Depth wise)

We have discussed flattening of a multi-level linked list where nodes have two pointers down and next. In the previous post, we flattened the linked list level wise. How to flatten a linked list when we always need to process down pointer before next at every node.

```
Input:
1 - 2 - 3 - 4
    |
    7 -  8 - 10 - 12
    |    |    |
    9    16   11
    |    |
    14   17 - 18 - 19 - 20
    |                   |
    15 - 23             21
         |
         24

Output:
Linked List to be flattened to
1 - 2 - 7 - 9 - 14 - 15 - 23 - 24 - 8
 - 16 - 17 - 18 - 19 - 20 - 21 - 10 -
11 - 12 - 3 - 4
Note : 9 appears before 8 (When we are
at a node, we process down pointer before
right pointer)
```

Source : Oracle Interview

If we take a closer look, we can notice that this problem is similar to tree to linked list conversion. We recursively flatten a linked list with following steps.

1) If node is NULL, return NULL.
2) Store next node of current node (used in step 4).
3) Recursively flatten down list. While flattening, keep track of last visited node, so that the next list can be linked after it.
4) Recursively flatten next list (we get the next list from pointer stored in step 2) and attach it after last visited node.

Below is C++ implementation of above idea.

```
// C++ program to flatten a multilevel linked list
#include <bits/stdc++.h>
using namespace std;

// A Linked List Node
struct Node
{
    int data;
    struct Node *next;
    struct Node *down;
};

// Flattens a multi-level linked list depth wise
Node* flattenList(Node* node)
{
    // Base case
    if (node == NULL)
        return NULL;

    // To keep track of last visited node
    // (NOTE: This is static)
    static Node *last;
    last = node;

    // Store next pointer
```

```
    Node *next = node->next;

    // If down list exists, process it first
    // Add down list as next of current node
    if (node->down)
       node->next = flattenList(node->down);

    // If next exists, add it after the next
    // of last added node
    if (next)
       last->next = flattenList(next);

    return node;
}

// Utility method to print a linked list
void printFlattenNodes(Node* head)
{
    while (head)
    {
        printf("%d ", head->data);
        head = head->next;
    }
}

// Utility function to create a new node
Node* newNode(int new_data)
{
    Node* new_node = new Node;
    new_node->data = new_data;
    new_node->next = new_node->down = NULL;
    return new_node;
}

// Driver code
int main()
{
    // Creating above example list
    Node* head = newNode(1);
    head->next = newNode(2);
    head->next->next = newNode(3);
    head->next->next->next = newNode(4);
    head->next->down = newNode(7);
    head->next->down->down = newNode(9);
    head->next->down->down->down = newNode(14);
    head->next->down->down->down->down
                                    = newNode(15);
    head->next->down->down->down->down->next
                                    = newNode(23);
    head->next->down->down->down->down->next->down
                                     = newNode(24);
    head->next->down->next = newNode(8);
    head->next->down->next->down = newNode(16);
    head->next->down->next->down->down = newNode(17);
    head->next->down->next->down->down->next
                                     = newNode(18);
    head->next->down->next->down->down->next->next
                                     = newNode(19);
    head->next->down->next->down->down->next->next->next
                                     = newNode(20);
    head->next->down->next->down->down->next->next->next->down
                                     = newNode(21);
    head->next->down->next->next = newNode(10);
    head->next->down->next->next->down = newNode(11);

    head->next->down->next->next->next = newNode(12);

    // Flatten list and print modified list
    head = flattenList(head);
    printFlattenNodes(head);

    return 0;
}
```

```
}
```

Output:

```
1 2 7 9 14 15 23 24 8 16 17 18 19 20 21 10 11 12 3 4
```