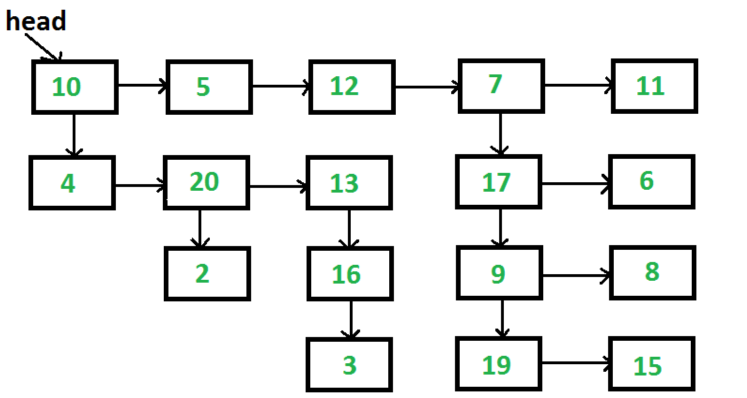
Flatten a multilevel linked list

Given a linked list where in addition to the next pointer, each node has a child pointer, which may or may not point to a separate list. These child lists may have one or more children of their own, and so on, to produce a multilevel data structure, as shown in below figure.You are given the head of the first level of the list. Flatten the list so that all the nodes appear in a single-level linked list. You need to flatten the list in way that all nodes at first level should come first, then nodes of second level, and so on.

Each node is a C struct with the following definition.

```
struct list
{
    int data;
    struct list *next;
    struct list *child;
};
```



The above list should be converted to 10->5->12->7->11->4->20->13->17->6->2->16->9->8->3->19->15

The problem clearly say that we need to flatten level by level. The idea of solution is, we start from first level, process all nodes one by one, if a node has a child, then we append the child at the end of list, otherwise we don’t do anything. After the first level is processed, all next level nodes will be appended after first level. Same process is followed for the appended nodes.

```
1) Take "cur" pointer, which will point to head of the fist level of the list
2) Take "tail" pointer, which will point to end of the first level of the list
3) Repeat the below procedure while "curr" is not NULL.
    I) if current node has a child then
 a) append this new child list to the "tail"
  tail->next = cur->child
 b) find the last node of new child list and update "tail"
  tmp = cur->child;
  while (tmp->next != NULL)
   tmp = tmp->next;
  tail = tmp;
    II) move to the next node. i.e. cur = cur->next
```

Following is the implementation of the above algorithm.

C

```
// Program to flatten list with next and child pointers
#include <stdio.h>
#include <stdlib.h>

// Macro to find number of elements in array
#define SIZE(arr) (sizeof(arr)/sizeof(arr[0]))
```

```c
// A linked list node has data, next pointer and child pointer
struct node
{
    int data;
    struct node *next;
    struct node *child;
};

// A utility function to create a linked list with n nodes. The data
// of nodes is taken from arr[].  All child pointers are set as NULL
struct node *createList(int *arr, int n)
{
    struct node *head = NULL;
    struct node *p;

    int i;
    for (i = 0; i < n; ++i) {
        if (head == NULL)
            head = p = (struct node *)malloc(sizeof(*p));
        else {
            p->next = (struct node *)malloc(sizeof(*p));
            p = p->next;
        }
        p->data = arr[i];
        p->next = p->child = NULL;
    }
    return head;
}

// A utility function to print all nodes of a linked list
void printList(struct node *head)
{
    while (head != NULL) {
        printf("%d ", head->data);
        head = head->next;
    }
    printf("\n");
}

// This function creates the input list.  The created list is same
// as shown in the above figure
struct node *createList(void)
{
    int arr1[] = {10, 5, 12, 7, 11};
    int arr2[] = {4, 20, 13};
    int arr3[] = {17, 6};
    int arr4[] = {9, 8};
    int arr5[] = {19, 15};
    int arr6[] = {2};
    int arr7[] = {16};
    int arr8[] = {3};

    /* create 8 linked lists */
    struct node *head1 = createList(arr1, SIZE(arr1));
    struct node *head2 = createList(arr2, SIZE(arr2));
    struct node *head3 = createList(arr3, SIZE(arr3));
    struct node *head4 = createList(arr4, SIZE(arr4));
    struct node *head5 = createList(arr5, SIZE(arr5));
    struct node *head6 = createList(arr6, SIZE(arr6));
    struct node *head7 = createList(arr7, SIZE(arr7));
    struct node *head8 = createList(arr8, SIZE(arr8));


    /* modify child pointers to create the list shown above */
    head1->child = head2;
    head1->next->next->next->child = head3;
    head3->child = head4;
    head4->child = head5;
    head2->next->child = head6;
    head2->next->next->child = head7;
    head7->child = head8;
    /* Return head pointer of first linked list.  Note that all nodes are
       reachable from head1 */
    return head1;
}

/* The main function that flattens a multilevel linked list */
void flattenList(struct node *head)
{
    /*Base case*/
    if (head == NULL)
       return;

    struct node *tmp;

    /* Find tail node of first level linked list */
    struct node *tail = head;
    while (tail->next != NULL)
        tail = tail->next;

    // One by one traverse through all nodes of first level
    // linked list till we reach the tail node
    struct node *cur = head;
    while (cur != tail)
    {
        // If current node has a child
        if (cur->child)
        {
            // then append the child at the end of current list
            tail->next = cur->child;

            // and update the tail to new last node
            tmp = cur->child;
            while (tmp->next)
                tmp = tmp->next;
            tail = tmp;
        }

        // Change current node
        cur = cur->next;
    }
}

// A driver program to test above functions
int main(void)
{
    struct node *head = NULL;
    head = createList();
    flattenList(head);
    printList(head);
    return 0;
}
```

Java

```
// Java program to flatten linked list with next and child pointers

class LinkedList {

    static Node head;

    class Node {

        int data;
        Node next, child;

        Node(int d) {
            data = d;
            next = child = null;
        }
    }
```

```
// A utility function to create a linked list with n nodes. The data
// of nodes is taken from arr[].  All child pointers are set as NULL
Node createList(int arr[], int n) {
    Node node = null;
    Node p = null;

    int i;
    for (i = 0; i < n; ++i) {
        if (node == null) {
            node = p = new Node(arr[i]);
        } else {
            p.next = new Node(arr[i]);
            p = p.next;
        }
        p.next = p.child = null;
    }
    return node;
}

// A utility function to print all nodes of a linked list
void printList(Node node) {
    while (node != null) {
        System.out.print(node.data + " ");
        node = node.next;
    }
    System.out.println("");
}

Node createList() {
    int arr1[] = new int[]{10, 5, 12, 7, 11};
    int arr2[] = new int[]{4, 20, 13};
    int arr3[] = new int[]{17, 6};
    int arr4[] = new int[]{9, 8};
    int arr5[] = new int[]{19, 15};
    int arr6[] = new int[]{2};
    int arr7[] = new int[]{16};
    int arr8[] = new int[]{3};

    /* create 8 linked lists */
    Node head1 = createList(arr1, arr1.length);
    Node head2 = createList(arr2, arr2.length);
    Node head3 = createList(arr3, arr3.length);
    Node head4 = createList(arr4, arr4.length);
    Node head5 = createList(arr5, arr5.length);
    Node head6 = createList(arr6, arr6.length);
    Node head7 = createList(arr7, arr7.length);
    Node head8 = createList(arr8, arr8.length);

    /* modify child pointers to create the list shown above */
    head1.child = head2;
    head1.next.next.next.child = head3;
    head3.child = head4;
    head4.child = head5;
    head2.next.child = head6;
    head2.next.next.child = head7;
    head7.child = head8;

    /* Return head pointer of first linked list.  Note that all nodes are
     reachable from head1 */
    return head1;
}

/* The main function that flattens a multilevel linked list */
void flattenList(Node node) {

    /*Base case*/
    if (node == null) {
        return;
    }

    Node tmp = null;
```

```
        /* Find tail node of first level linked list */
        Node tail = node;
        while (tail.next != null) {
            tail = tail.next;
        }

        // One by one traverse through all nodes of first level
        // linked list till we reach the tail node
        Node cur = node;
        while (cur != tail) {

            // If current node has a child
            if (cur.child != null) {

                // then append the child at the end of current list
                tail.next = cur.child;

                // and update the tail to new last node
                tmp = cur.child;
                while (tmp.next != null) {
                    tmp = tmp.next;
                }
                tail = tmp;
            }

            // Change current node
            cur = cur.next;
        }
    }

    public static void main(String[] args) {
        LinkedList list = new LinkedList();
        head = list.createList();
        list.flattenList(head);
        list.printList(head);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
10 5 12 7 11 4 20 13 17 6 2 16 9 8 3 19 15
```

Time Complexity: Since every node is visited at most twice, the time complexity is O(n) where n is the number of nodes in given linked list.