

Merge two sorted linked lists

Write a SortedMerge() function that takes two lists, each of which is sorted in increasing order, and merges the two together into one list which is in increasing order. SortedMerge() should return the new list. The new list should be made by splicing together the nodes of the first two lists.

For example if the first linked list a is 5->10->15 and the other linked list b is 2->3->20, then SortedMerge() should return a pointer to the head node of the merged list 2->3->5->10->15->20.

There are many cases to deal with: either 'a' or 'b' may be empty, during processing either 'a' or 'b' may run out first, and finally there's the problem of starting the result list empty, and building it up while going through 'a' and 'b'.

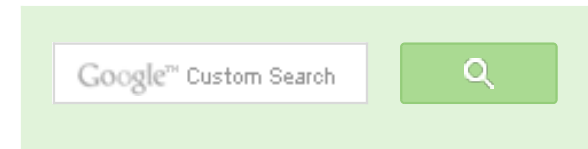
Method 1 (Using Dummy Nodes)

The strategy here uses a temporary dummy node as the start of the result list. The pointer Tail always points to the last node in the result list, so appending new nodes is easy.

The dummy node gives tail something to point to initially when the result list is empty. This dummy node is efficient, since it is only temporary, and it is allocated in the stack. The loop proceeds, removing one node from either 'a' or 'b', and adding it to tail. When we are done, the result is in dummy.next.

```
/*Program to alternatively split a linked list into two halves */
#include<stdio.h>
#include<stdlib.h>
#include<assert.h>

/* Link list node */
struct node
{
```



GeeksforGeeks



53,528 people like [GeeksforGeeks](#).



[Interview Experiences](#)

[Advanced Data Structures](#)

[Dynamic Programming](#)

[Greedy Algorithms](#)

[Backtracking](#)

[Pattern Searching](#)

[Divide & Conquer](#)

[Mathematical Algorithms](#)

[Recursion](#)

```

    int data;
    struct node* next;
};

/* pull off the front node of the source and put it in dest */
void MoveNode(struct node** destRef, struct node** sourceRef);

/* Takes two lists sorted in increasing order, and splices their nodes
struct node* SortedMerge(struct node* a, struct node* b)
{
    /* a dummy first node to hang the result on */
    struct node dummy;

    /* tail points to the last result node */
    struct node* tail = &dummy;

    /* so tail->next is the place to add new nodes
       to the result. */
    dummy.next = NULL;
    while(1)
    {
        if(a == NULL)
        {
            /* if either list runs out, use the other list */
            tail->next = b;
            break;
        }
        else if (b == NULL)
        {
            tail->next = a;
            break;
        }
        if (a->data <= b->data)
        {
            MoveNode(&(tail->next), &a);
        }
        else
        {
            MoveNode(&(tail->next), &b);
        }
        tail = tail->next;
    }
    return(dummy.next);
}

/* UTILITY FUNCTIONS */

```



Popular Posts

All permutations of a given string

Memory Layout of C Programs

Understanding "extern" keyword in C

Median of two sorted arrays

Tree traversal without recursion and without stack!

Structure Member Alignment, Padding and Data Packing

Intersection point of two Linked Lists

Lowest Common Ancestor in a BST.

Check if a binary tree is BST or not

Sorted Linked List to Balanced BST

```
/*MoveNode() function takes the node from the front of the source, and  
It is an error to call this with the source list empty.
```

```
Before calling MoveNode():
```

```
source == {1, 2, 3}
```

```
dest == {1, 2, 3}
```

```
After calling MoveNode():
```

```
source == {2, 3}
```

```
dest == {1, 1, 2, 3}
```

```
*/
```

```
void MoveNode(struct node** destRef, struct node** sourceRef)
```

```
{
```

```
/* the front source node */
```

```
struct node* newNode = *sourceRef;
```

```
assert(newNode != NULL);
```

```
/* Advance the source pointer */
```

```
*sourceRef = newNode->next;
```

```
/* Link the old dest off the new node */
```

```
newNode->next = *destRef;
```

```
/* Move dest to point to the new node */
```

```
*destRef = newNode;
```

```
}
```

```
/* Function to insert a node at the beginning of the linked list */
```

```
void push(struct node** head_ref, int new_data)
```

```
{
```

```
/* allocate node */
```

```
struct node* new_node =  
    (struct node*) malloc(sizeof(struct node));
```

```
/* put in the data */
```

```
new_node->data = new_data;
```

```
/* link the old list off the new node */
```

```
new_node->next = (*head_ref);
```

```
/* move the head to point to the new node */
```

```
(*head_ref) = new_node;
```

```
}
```

```
/* Function to print nodes in a given linked list */
```

```
void printList(struct node *node)
```



Recent Comments

Abhi You live US or India?

Google (Mountain View) interview · 50 minutes ago

Aman Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 1 hour ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 1 hour ago

Sanjay Agarwal bool

tree::Root_to_leaf_path_given_sum(tree...

Root to leaf path sum equal to a given number · 1 hour ago

GOPI GOPINATH @admin Highlight this sentence "We can easily...

Count trailing zeroes in factorial of a number · 1 hour ago

newCoder3006 If the array contains negative numbers also. We...

Find subarray with given sum · 2 hours ago

AdChoices

[▶ Linked List](#)

[▶ Programming C++](#)

[▶ C++ Merge List](#)

AdChoices

[▶ Java Array](#)

```

{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Drier program to test above functions*/
int main()
{
    /* Start with the empty list */
    struct node* res = NULL;
    struct node* a = NULL;
    struct node* b = NULL;

    /* Let us create two sorted linked lists to test the functions
       Created lists shall be a: 5->10->15, b: 2->3->20 */
    push(&a, 15);
    push(&a, 10);
    push(&a, 5);

    push(&b, 20);
    push(&b, 3);
    push(&b, 2);

    /* Remove duplicates from linked list */
    res = SortedMerge(a, b);

    printf("\n Merged Linked List is: \n");
    printList(res);

    getchar();
    return 0;
}

```

Method 2 (Using Local References)

This solution is structurally very similar to the above, but it avoids using a dummy node. Instead, it maintains a struct node** pointer, lastPtrRef, that always points to the last pointer of the result list. This solves the same case that the dummy node did — dealing with the result list when it is empty. If you are trying to build up a list at its tail, either the dummy node or the struct node** “reference” strategy can be used (see Section 1 for details).


```

struct node* SortedMerge(struct node* a, struct node* b)

```

► [NODE](#)

► [Reverse Merge](#)

AdChoices 

► [Reverse Merge](#)

► [A Merge](#)

► [Merge In](#)

```
{
    struct node* result = NULL;

    /* point to the last result pointer */
    struct node** lastPtrRef = &result;

    while(1)
    {
        if (a == NULL)
        {
            *lastPtrRef = b;
            break;
        }
        else if (b==NULL)
        {
            *lastPtrRef = a;
            break;
        }
        if(a->data <= b->data)
        {
            MoveNode(lastPtrRef, &a);
        }
        else
        {
            MoveNode(lastPtrRef, &b);
        }

        /* tricky: advance to point to the next ".next" field */
        lastPtrRef = &((*lastPtrRef)->next);
    }
    return(result);
}
```

Method 3 (Using Recursion)

Merge is one of those nice recursive problems where the recursive solution code is much cleaner than the iterative code. You probably wouldn't want to use the recursive version for production code however, because it will use stack space which is proportional to the length of the lists.

```
struct node* SortedMerge(struct node* a, struct node* b)
{
    struct node* result = NULL;

    /* Base cases */
    if (a == NULL)
```

```

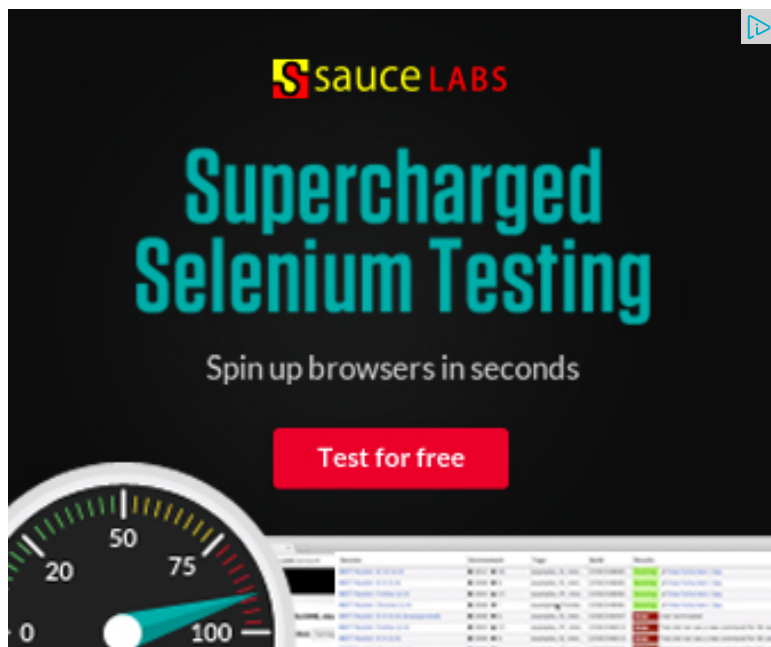
    return(b);
else if (b==NULL)
    return(a);

/* Pick either a or b, and recur */
if (a->data <= b->data)
{
    result = a;
    result->next = SortedMerge(a->next, b);
}
else
{
    result = b;
    result->next = SortedMerge(a, b->next);
}
return(result);
}

```

Source: <http://cslibrary.stanford.edu/105/LinkedListProblems.pdf>

Please write comments if you find the above code/algorithm incorrect, or find better ways to solve the same problem.



Related Topics:

- Given a linked list, reverse alternate nodes and append at the end
- Pairwise swap elements of a given linked list by changing links
- Self Organizing List | Set 1 (Introduction)
- Merge a linked list into another linked list at alternate positions
- QuickSort on Singly Linked List
- Delete N nodes after M nodes of a linked list
- Design a stack with operations on middle element
- Swap Kth node from beginning with Kth node from end in a Linked List



13



Tweet

0



0

Writing code in comment? Please use ideone.com and share the link here.

@geeksforgeeks, **Some rights reserved**

Contact Us!

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team