Given a linked list, reverse alternate nodes and append at the end

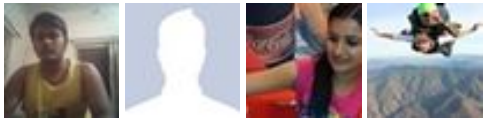
Given a linked list, reverse alternate nodes and append them to end of list. Extra allowed space is O(1)
Examples

```
Input List:  1->2->3->4->5->6
Output List: 1->3->5->6->4->2

Input List:  12->14->16->18->20
Output List: 12->16->20->18->14
```

We strongly recommend to minimize the browser and try this yourself first.

The idea is to maintain two linked lists, one list of all odd positioned nodes (1, 3, 5 in above example) and other list of all even positioned nodes (6, 4 and 2 in above example). Following are detailed steps.
1) Traverse the given linked list which is considered as odd list. Do following for every visited node.
……**a)** If the node is even node, remove it from odd list and add it to the front of even node list. Nodes are added at front to keep the reverse order.
2) Append the even node list at the end of odd node list.

```
#include<stdio.h>
#include<stdlib.h>

/* A linked list node */
struct node
```

Interview Experiences
Advanced Data Structures
Dynamic Programming
Greedy Algorithms
Backtracking
Pattern Searching
Divide & Conquer
Mathematical Algorithms
Recursion

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

```
/* Function to reverse all even positioned node and append at the end
   odd is the head node of given linked list */
void rearrange(struct node *odd)
{
    // If linked list has less than 3 nodes, no change is required
    if (odd == NULL || odd->next == NULL || odd->next->next == NULL)
        return;

    // even points to the beginning of even list
    struct node *even = odd->next;

    // Remove the first even node
    odd->next = odd->next->next;

    // odd points to next node in odd list
    odd = odd->next;

    // Set terminator for even list
    even->next = NULL;

    // Traverse the  list
    while (odd && odd->next)
    {
       // Store the next node in odd list
       struct node *temp = odd->next->next;

       // Link the next even node at the beginning of even list
       odd->next->next = even;
       even = odd->next;

       // Remove the even node from middle
       odd->next = temp;

       // Move odd to the next odd node
       if (temp != NULL)
         odd = temp;
    }

    // Append the even list at the end of odd list
    odd->next = even;
}
```

Geometric Algorithms

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

```c
/* Function to add a node at the beginning of Linked List */
void push(struct node** head_ref, int new_data)
{
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
    new_node->data  = new_data;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}

/* Function to print nodes in a given linked list */
void printList(struct node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}

/* Druver program to test above function */
int main()
{
    struct node *start = NULL;

    /* The constructed linked list is:
     1->2->3->4->5->6->7 */
    push(&start, 7);
    push(&start, 6);
    push(&start, 5);
    push(&start, 4);
    push(&start, 3);
    push(&start, 2);
    push(&start, 1);

    printf("\n Linked list before calling  rearrange() ");
    printList(start);

    rearrange(start);

    printf("\n Linked list after calling  rearrange() ");
    printList(start);

    return 0;
}
```

Output:

```
Linked list before calling  rearrange() 1 2 3 4 5 6 7
Linked list after calling  rearrange()  1 3 5 7 6 4 2
```

Time Complexity: The above code simply traverses the given linked list. So time complexity is O(n)

Auxiliary Space: O(1)

This article is contributed by **Aman Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Tpoics:

- 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

Recent Comments

- QuickSort on Doubly Linked List

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

28 Comments GeeksforGeeks

Sort by Newest

santosh gupta • 5 days ago

```
void rearrange(struct node *odd)
{

if(odd==NULL||odd->next==NULL||odd->next->next==NULL){

return;
}
struct node *even, *temp,*temp1,*par;
even=odd->next;

odd->next=odd->next->next;
odd=odd->next;

even->next=NULL;

while(odd&&odd->next){
temp1=odd->next->next;

temp=odd->next;
temp->next=even;
```

see more

sandeep • 11 days ago

```
void rearrange(struct node *head)
{
if(head==NULL)
return;
struct node *p1=head;
struct node *p2=head->next;
struct node *curr=NULL;
if(p2!=NULL)
{
curr=p2->next;
p2->next=NULL;
}
while(p1!=NULL && curr!=NULL)
{
p1->next=curr;
p1=curr;

if(p1->next!=NULL)
```

see more

Ankit Jain • 25 days ago

```
#include<stdio.h>
#include<stdlib.h>
struct Node
{
int data;
struct Node *next;
```

```
};

struct Node * insertLinked(struct Node *head,int data)
{
struct Node *temp=head;
if(temp==NULL)
{
temp=(struct Node*)malloc(sizeof(struct Node));
temp->data=data;
temp->next=NULL;
return temp;
}
```

see more

• Reply • Share ›

chandan • a month ago

Can anyone explain me how auxillary space is o(1)

3 • Reply • Share ›

Harsha • a month ago

```
Stack<locallinkedlist.node> oddNodes = new Stack<locallinkedlist.node>(null)
LocalLinkedList.Node current = LL.getHead();
while(current.getNext()!=null){
oddNodes.push(current.getNext());
current.setNext(current.getNext().getNext());
if(current.getNext()!=null)
current = current.getNext();
}
while(oddNodes.getTop()!=null){
current.setNext(oddNodes.pop());
current = current.getNext();
```

```
}
return LL;
```

Reply • Share ›

Guest • a month ago

evenodd pos

Reply • Share ›

Sumitgolusagar • a month ago

```
node *alt_node_reverse(node *p){
node *head,*q,*temp;
head=q=p;
temp=NULL;
while(p && p->next){
q=p->next;
if(q->next!=NULL){
p->next=q->next;
p=p->next;
}
else
p->next=NULL;
q->next=temp;
temp=q;
}
p->next=temp;
return head;
}
```

Reply • Share ›

xxx Sumitgolusagar • a month ago

this solution is wrong

Reply • Share ›

Aniruddha • 2 months ago

I found the solution very easy to understand and implement. I, like many others pluck the nodes at even positions and add them after the last node in the origi

```
void LinkedListProblems::ReverseAlternateNodesAndAppendAtTheEnd()
{
LinkedList list;
unsigned int listSize = 11;

LinkedList::Populate(list, listSize);
cout << "Printing ... ";
list.Print();

const unsigned int length = list.Length();
if (length < 3)
{
return;
}

// Even number of nodes
```

see more

• Reply • Share ›

Aman Agarwal • 2 months ago

```
struct node{
int data;
struct node *next;
}*ptr,*start,*newnode;

void appendodd()
{
```

```
int c=1;//counter to find even or odd no of item
ptr=start;
struct node *last,*first,*prev;
while(ptr->next!=NULL)
{
ptr=ptr->next;
}
last=ptr; //last point to the last element of the original linklist
ptr=start;
while(ptr!=last)
{
```

see more

Reply • Share ›

RGuest • 2 months ago

```
// A linked list node
struct node
{
int data;
struct node *next;
};

/* Function to insert a node at the beginning */
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);
```

see more

Reply • Share ›

RGuest • 2 months ago

```
void push1(struct node ** head_ref, struct node *new_node)
{
/* 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 skip M nodes and then delete N nodes of the linked list.
void rearrange(struct node *head, int M, int N)
{
struct node *curr = head, *t;
int count;

struct node *result = NULL;
// The main loop that traverses through the whole list
while (curr)
{
// Skip M nodes
```

see more

Reply • Share ›

ajay • 3 months ago

```
void rearrange(struct node* head)
{
struct node* temp = NULL;
```

struct node* temp = NULL;

while(head and head->next)
{
struct node* pre = head->next;
head->next = head->next->next;
if(head->next)
head = head->next;
pre->next = temp;
temp = pre;
}
head->next = temp;

}

Reply • Share ›

Sudhanshu • 3 months ago

I used a rather simple approach to this problem. no even or odd ..

Reply • Share ›

Sudhanshu • 3 months ago

void reArrange(struct node* start)

{

struct node* a,*r,*i,*s,*x;

for(i=start;i->next!=NULL ;i=i->next);

a=i;

for(i=start;i!=a && i->next!=a;i=i->next){

r=i->next;

```
s=r->next;

i->next=s;

x=a->next;

a->next=r;

r->next=x;

}

}
```

Reply • Share ›

hxgxs1 • 3 months ago

http://ideone.com/T4Bu2X

Traverse the list pluck out the even nodes and insert them at the last node's n
e.g
input:: 1 2 3 4 5 6
after 1st loop execution : 1 3 4 5 6 2
after 2nd loop execution : 1 3 5 6 4 2
That's it.

1 Reply • Share ›

suresh kumar mahawar • 3 months ago

can it implement recursively?

Reply • Share ›

saurabh • 3 months ago

Why not just traverse, and unlink and insert the even nodes after the original ta
Here are the relevant functions:

saurabh saurabh • 3 months ago

```
void list::revappend()//public calling function
{
oldtail=tail;
walkthrough(head);
return;
}

void list::walkthrough(Node* r)//private recursive function to walkthroug
{
if(r==NULL)
return;
if(r==oldtail ||r->next==oldtail)
return;
Node* nextnode=r->next;
if(r->next)
r->next=r->next->next;
insert_after(nextnode);
walkthrough(r->next);
```

see more

Karthikeyan Mahadevan • 3 months ago

```
void ReverseAltNode(Node* pln)

{

Node* pTempNodeList = NULL;

if(pln == NULL)
```

```
return;

bool isAlternate = false;

Node* nodeItr = pIn;

while(nodeItr && nodeItr->next)

{

if(isAlternate)

{
```

see more

Reply • Share ›

geekykid • 3 months ago

shouldn't output be 1 3 5 7 6 4 2 instead of 1 4 6 7 5 3 2

2 Reply • Share ›

GeeksforGeeks Mod geekykid • 3 months ago

We have fixed the code.

Reply • Share ›

geekykid geekykid • 3 months ago

there is flaw in the code. 'odd' pointer is not changed before entering th
please check

Reply • Share ›

GeeksforGeeks Mod geekykid • 3 months ago

Thanks for pointing this out. We have fixed the problem.

Reply • Share ›

Aditya • 3 months ago

```
public Node reverseInAlternate(Node head){
Node curr = head;
if (curr==null || curr.next==null || curr.next.next==null) return head;
Node prevOdd = null;
Node prevEven = null;
boolean operation=true;
Node tmp;
while (curr!=null){
if (operation==true){
tmp = curr.next;
if (curr.next!=null)
curr.next = curr.next.next;
prevEven = curr;
curr = tmp;
}
else{
tmp = curr.next;
curr.next = prevOdd;
```

see more

^ | v • Reply • Share ›

nani ➦ Aditya • 3 months ago

the output should be Linked list before calling rearrange() 1 2 3 4 5 6 7
Linked list after calling rearrange() 1 4 6 7 5 3 2::

1 3 5 7 6 4 2 ... instead of 14 6 7 5 3 2

^ | v • Reply • Share ›

Aditya ➦ nani • 3 months ago

Is the code above giving 1 4 6 7 5 3 2?

1 • Reply • Share ›

Aditya Aditya • 3 months ago

I tested my code and it gives 1 3 5 7 6 4 2 for the input 1

• Reply • Share ›

Subscribe Add Disqus to your site