Delete alternate nodes of a Linked List

Given a Singly Linked List, starting from the second node delete all alternate nodes of it. For example, if the given linked list is 1->2->3->4->5 then your function should convert it to 1->3->5, and if the given linked list is 1->2->3->4 then convert it to 1->3.

Method 1 (Iterative)
Keep track of previous of the node to be deleted. First change the next link of previous node and then free the memory allocated for the node.

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

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

/* deletes alternate nodes of a list starting with head */
void deleteAlt(struct node *head)
{
  if (head == NULL)
    return;

   /* Initialize prev and node to be deleted */
  struct node *prev = head;
  struct node *node = head->next;

  while (prev != NULL && node != NULL)
  {
     /* Change next link of previous node */
     prev->next = node->next;

     /* Free memory */
     free(node);

     /* Update prev and node */
     prev = prev->next;
     if(prev != NULL)
        node = prev->next;
  }
}

/* UTILITY FUNCTIONS TO TEST fun1() and fun2() */
/* Given a reference (pointer to pointer) to the head
  of a list and an int, push a new node on the front
  of the 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)
{
  while(node != NULL)
  {
    printf("%d ", node->data);
    node = node->next;
  }
}

/* Drier program to test above functions */
int main()
{
  int arr[100];

  /* Start with the empty list */
  struct node* head = NULL;

  /* Using push() to construct below list
    1->2->3->4->5  */
  push(&head, 5);
  push(&head, 4);
  push(&head, 3);
  push(&head, 2);
  push(&head, 1);

  printf("\n List before calling deleteAlt() ");
  printList(head);

  deleteAlt(head);

  printf("\n List after calling deleteAlt() ");
  printList(head);

  getchar();
  return 0;
}
```

Time Complexity: O(n) where n is the number of nodes in the given Linked List.

Method 2 (Recursive)
Recursive code uses the same approach as method 1. The recursive coed is simple and short, but causes O(n) recursive function calls for a linked list of size n.

```
/* deletes alternate nodes of a list starting with head */
void deleteAlt(struct node *head)
{
  if (head == NULL)
    return;

  struct node *node = head->next;

  if (node == NULL)
```

```
    return;

  /* Change the next link of head */
  head->next = node->next;

  /* free memory allocated for node */
  free(node);

  /* Recursively call for the new next of head */
  deleteAlt(head->next);
}
```

Time Complexity: O(n)

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

Related Topics:

- Clone a linked list with next and random pointer | Set 2
- Given a linked list of line segments, remove middle points
- Construct a Maximum Sum Linked List out of two Sorted Linked Lists having some Common nodes
- 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

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

http://ideone.com/cQ1ISH

• Reply • Share ›

TheNewGuy • 23 days ago

Sorry, if I am wrong but I am new here. Shouldn't the parameters be node ** head for delAlt? We are changing the linked list here right? Otherwise the above function wouldn't change anything in the original LL.

• Reply • Share ›

Aditya Goel ➦ TheNewGuy • 14 days ago

Yes, we are changing the list here but we are not changing head pointer. We are just changing head->next not head itself. So ** is not needed here.
Hope you understood the difference.

• Reply • Share ›

Utkarsh Mishra • a month ago

```
node* del(node *head)
{
node *temp,*loop=head;
while(loop!=NULL && loop->link!=NULL)
{
temp=loop->link;
loop->link=temp->link;
free(temp);
loop=loop->link;
}

return head;
}
```

• Reply • Share ›

thevagabond85 • 2 months ago

a more succinct code for ITERATIVE version which uses less variable and condition check is :

```
typedef struct node* Node;

void deleteAlt(Node head){

Node temp;
while( head && (head->next)){
temp = head->next;
head->next = (head->next)->next;
free(temp);
head = head->next;
}
```

```
}
```

and for RECURSIVE version

```
void deleteAltRec(Node head){
Node temp;
if( head != NULL && head->next!=NULL) {
temp = head->next;
head->next= temp->next;
free(temp);
deleteAltRec(head->next);
}
}
```

1 • Reply • Share ›

thevagabond85 ➚ thevagabond85 • 2 months ago

tested for tree - NULL, only one, only 2, and so on.... successfully.

• Reply • Share ›

Abhinav Gupta • 2 months ago

Please find an elegant solution. Let know if there are any bugs in the code.

http://ideone.com/pPfyjd

• Reply • Share ›

Naval • 3 months ago

iterative version using 2 pointer 1 for keep track of the list and 2nd for free the memory block

```
void delete(struct node *curr)
{
struct node *prev;
if(curr==NULL && curr->next==NULL)
return ;
while(curr!=NULL && curr->next!=NULL)
{
prev=curr;
curr=curr->next;
prev->next=curr->next;
curr->next=NULL;
free(curr);
curr=prev->next;
}
}
```

• Reply • Share ›

vamshi reddy • 3 months ago

http://coliru.stacked-crooked....

• Reply • Share ›

Murthy • 5 months ago

```
Node node = head;
while (node != null && node.nextNode != null) {
node.nextNode = node.nextNode.nextNode;
node = node.nextNode;
}
```

• Reply • Share ›

Ankit Sablok • 7 months ago

I think this question can be solved without using 2 pointers current and previous as well isn't it - here is the code to support my claim

```
void LinkedList :: deleteAlternateNodes(){

if(this->head == N)
return;

Node* temp = this->head;

while(temp != N){
if(temp->next == N)
break;

temp->next = temp->next->next;
temp = temp->next;
}
}
```

• Reply • Share ›

Giridhar ↗ Ankit Sablok • 6 months ago

Approach is correct. However it leads to memory leaks in C.
in C how do you free the node with this approach Ankit ?

1 • Reply • Share ›

Batman_ • 7 months ago

```
delete(node *prev,node *curr)
{

prev->next=curr->next;

free(curr);
```

```
}

removeAlterNodes(node *prev,node *curr)
{

if(curr==NULL)

return;

delete(prev,curr);

removeAlterUtil(prev->next,prev->next->next);

}
```

• Reply • Share ›

Kim Jong-il • 7 months ago

Simple Iterative Implementation:

```
void delete_alternate(struct node *head)
{
struct node *temp=head,*del;
while(temp && temp->next)
{
del= temp->next;
temp->next = del->next;
free(del);
temp=temp->next;
}
}
```

1 • Reply • Share ›

ajayv • 7 months ago

```
void deleteAlt(struct node *head)
{
if(head == NULL || head ->next == NULL) return;

struct node* curr = head;
struct node* eve = curr->next;
if(curr && eve){
curr->next = eve->next;
free(eve);
curr = curr->next;
deleteAlt(curr);
}
}
```

1 • Reply • Share ›

ashishj jaiswal ajayv • 3 months ago
You code is causing Segmentation fault....check it please
• Reply • Share ›

avinash ajayv • 7 months ago
are you sure, you are not getting segmentation fault with this code?
• Reply • Share ›

DS+Algo=Placement • 8 months ago
Is this recursive approach called tail recursive?
• Reply • Share ›

Jun DS+Algo=Placement • 8 months ago
yup
• Reply • Share ›

<HoldOnLife!#> DS+Algo=Placement • 8 months ago
yes
• Reply • Share ›

Vinayak • 9 months ago
My code is not working as expected. the same original list is being printed.
can someone please check? Thanks
Here is the Ideone link-

Here's the function-
http://ideone.com/D9jhp4

```
struct node*delete_alternate(struct node*start ){

struct node*temp, *p;

p=start;

if(p->link!=NULL)return start;

while(p!=NULL){

temp=p->link;

p->link=temp->link;

p=temp;

free(temp);
```

```
if(p->link!=NULL)p=p->link;

}return start;

}
```

• Reply • Share ›

sonu431 • 9 months ago

this can be even more easier .....to get

```
void deleteAlt (struct node *head)
{
struct node*temp=head;
struct node*delnode, *savenode;

if (!head)
return;

while ( temp!=NULL && temp->next!=NULL)
{
delnode=temp->next;
savenode=delnode->next;
temp->next=savenode;
free(delnode);
temp=savenode;
}
}
```

• Reply • Share ›

Aswin Gokulachandran • 9 months ago

Java Implementation (Only the method)

```
-------------------------

/*

* Method 1 to delete the alternate elements in the list

*/

public void deleteAlternateNodes(){

currNode = head;

while(currNode.next!=null){

currNode.next = currNode.next.next;
```

```
if(currNode.next!=null)

currNode = currNode.next;
```

see more

• Reply • Share ›

AMIT JAMBOTKAR • a year ago

Java implementation is here

```
package deletealtrnatenodesoflinkedlist;

public class LinkedList<e> implements Cloneable {

Node<e> head = null;

// Adding at the End

class Node<t> {

T value;

Node<t> nextReference;

public Node(T value) {

this.value = value;

this.nextReference = null;
```

see more

• Reply • Share ›

Viswanath • a year ago

My implementation is better and shorter

1 • Reply • Share ›

Himanshu Dagar • a year ago

For combined code can refer to below link

http://ideone.com/Om2sOW

• Reply • Share ›

Shikha Gupta • a year ago

is anything wrong in the following code:

```
void delete_alternate_node(struct node **headref)
{
struct node *cur=*headref;
```

```
struct node *cur=headref;
struct node *temp=NULL;

while(cur!=NULL&&cur->link!=NULL)
{
temp=cur->link;
cur->link=cur->link->link;
free(temp);
cur=cur->link;
}
}
```

• Reply • Share ›

Kartik Nagpal ➚ Shikha Gupta • a year ago

http://ideone.com/u4fkZG

Here's a complete C++ code for the same, using your code for delete_alternate_node routine. Hope you will find it helpful.

• Reply • Share ›

Shikha Gupta ➚ Kartik Nagpal • a year ago

ya thanks it works fine..bt i havent chcked 4 all test cases

• Reply • Share ›

Kartik ➚ Shikha Gupta • a year ago

all the relavent test cases here would be: 1) empty list, 2) odd length list, and 3) even length list.

The above code passes all.

1 • Reply • Share ›

wakeup123 • 2 years ago

in method 1, why int arr[100]; in the driver portion of the code in the beginning of int main?

• Reply • Share ›

Adarsh • 2 years ago

```
 /* ========================================================================== */
/*                                                                            */
/*   Filename.c                                                               */
/*   (c) 2001 Author                                                          */
/*                                                                            */
/*   Description                                                              */
/*                                                                            */
/* ========================================================================== */
#include<stdio.h>
#include<stdlib.h>
```

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

void display(node **head_ref)
{
```

see more

• Reply • Share ›

vs • 2 years ago

For method 1
Inside the while loop
prev = prev->next; is not required
Suppose there are two nodes then
after during the first iteration of the loop
prev->next = node->next; sets the prev to NULL
Therefore, an error will occur while trying to access
next of prev.

Removing this line would make the code correct

• Reply • Share ›

Ankit vs • 2 years ago

Why do you say so ??!! when prev will get to NULL then simply the while loop will terminate....

• Reply • Share ›

yatharth.sharma • 2 years ago

```
node * recdel(node *t)
{
if(t==NULL) return NULL;
if(t->next==NULL)return t;
t->next=recdel(t->next->next);

}
```

• Reply • Share ›

Nikin Kumar Jain • 2 years ago

```
void delAlternateNode(node *sr)
{
        while(sr)
        {
                if(sr->next)
                {
                        node *temp = sr->next;
                        sr->next = sr->next->next;
                        delete temp;
                        sr = sr->next;
                }
                sr = sr->next;
        }
}
```

• Reply • Share ›

Arindam Sanyal • 2 years ago

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

struct node{
int info;
struct node *link;.
};

struct node * addtoempty(struct node *, int);.
struct node * addtoend(struct node *, int);.
void display(struct node *);.
struct node *delalt(struct node *);.

void main(){
clrscr();
struct node *start=NULL;.
int num, d;
printf("\n enter the number of nodes ");

see more

• Reply • Share ›

hARRY • 2 years ago

This complexity is o(n/2) not o(n) exactly...

• Reply • Share ›

learning • 3 years ago

@Sambasiva shoudnt the for loop condition be this as when there are odd no. of nodes then we are trying to free a null pointer in ur solution which is incorrect i guess.

```
void deleteAlternateNode(list l)
{
Node *temp;

for( ; l && l->next ; l = l->next)
{
temp = l->next;
l->next = temp->next;
free(temp);
}
}
```

• Reply • Share ›

learner • 3 years ago

```
while (prev != NULL && node != NULL)
 {
    /* Change next link of previous node */
    prev->next = node->next;

    /* Free memory */
    free(node);

    /* Update prev and node */
    prev = prev->next;
    if(prev != NULL) //statement 1
       node = prev->next;
  }
}
```

Why are we checking for prev!=NULL in statement 1. Is that necessary.If prev is null,then node will also be null which in turn fails to satisfy the condition in the while loop and thus the iteration stops right ?

1 • Reply • Share ›

Praveen ➚ learner • 3 years ago

It is necessary because if the linked list is of even length e.g. LL is 1->2->3->4, then in the second iteration of while loop prev is set to null but node is not set to null. So the condition is required if not program gives error as the while loop executes.

• Reply • Share ›

Venki • 4 years ago

Recursive method missing dual reference of head pointer. And similar modification is required when checking against its NULL-ness.

• Reply • Share ›

Bandicoot • 4 years ago

Once again, Sambasiva's soln trumps both the solns that you gave above in terms of simplicity and readability.

• Reply • Share ›

R.Srinivasan ➚ Bandicoot • 4 years ago

@Bandicoot
Sambasiva's program fails("runtime error") when the number of nodes is odd(1,3,etc), since temp=NULL(temp=l->next) when l becomes the last node and the illegal pointer Exception in "l->next=temp->next".(temp=NULL).

• Reply • Share ›

Bandicoot ➚ R.Srinivasan • 4 years ago

My bad. You are right. Sambasiva should have added a if(!temp) conditional before trying to access either temp or temp->next. Apart from that, his soln is good.

• Reply • Share ›

R.Srinivasan • 4 years ago

```
 void deletealtnode(struct node * head)
{
    struct node * current=head,*temp;
    while(current!=NULL && current->next!=NULL)
    {
           temp=current->next;
           current->next=temp->next;
           current=current->next;
           free(temp);
     }
}
```

• Reply • Share ›

Vinay Kumar • 4 years ago

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

```
}mynode;

void deleteAlt(mynode *head)
{
  /* Initialize prev and node to be deleted */
  mynode *current = head;
  mynode *nextPtr;

  while (current != NULL)
  {
     nextPtr = current->next;

     if(nextPtr)
     {
```

see more

• Reply • Share ›

Techiee • 5 years ago

My 2 cents:

```
void deleteAlternate (Node *n) {
   while (n && n->next) {
       Node *nxt_nxt = n->next->next;
       delete (n->next);
       n->next = nxt_nxt;
       n = nxt_nxt;
   }
}
```

1 • Reply • Share ›

<HoldOnLife!#> Techiee • 8 months ago

does it works with all testcases ? it seems so

• Reply • Share ›

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

Recent Comments

- Baba

  Traverse the tree in inorder fashion and insert...

  Populate Inorder Successor for all nodes · 54 minutes ago

- creeping_death

  Ruby solution, slightly different, easier to...

Longest Even Length Substring such that Sum of First and Second Half is same · 1 hour ago

- darkprotocol

  Anyone round-1 4th question?

  Amazon Interview | Set 121 (On-Campus for SDE-1) · 1 hour ago

- S.Nkm

  Would a simple answer involving the Unix epoch...

  Find number of days between two given dates · 1 hour ago

- Shomina

  Found Real Job INterview Questions Here...

  Flipkart Interview Experience | Set 20 (For SDE-II) · 2 hours ago

- Shimona

  Found Real Job INterview Questions here...

  Myntra Interview Experience | Set 4 (For Senior Software Engineer) · 2 hours ago