Write a function to delete a Linked List

Algorithm: Iterate through the linked list and delete all the nodes one by one. Main point here is not to access next of the current pointer if current pointer is deleted.

Implementation:

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

/* Link list node */
```

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

/* Function to delete the entire linked list */
void deleteList(struct node** head_ref)
{
   /* deref head_ref to get the real head */
   struct node* current = *head_ref;
   struct node* next;

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

   /* deref head_ref to affect the real head back
      in the caller. */
   *head_ref = NULL;
}

/* 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;
}

/* Drier program to test count function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    /* Use push() to construct below list
     1->12->1->4->1  */
    push(&head, 1);
```

```
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
    push(&head, 1);

    printf("\n Deleting linked list");
    deleteList(&head);

    printf("\n Linked list deleted");
    getchar();
}
```

Time Complexity: O(n)
Space Complexity: O(1)

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

Tags: Delete a Linked List, Linked Lists

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

Pawan Dwivedi • 2 months ago

Recursive solution

```
void deletelist(struct node *n)
{
if(n->next!=NULL)
{
deletelist(n->next);
}
```

printf("Node to be deleted: %d\n",n->data);
free(n);
}

1 • Reply • Share ›

NotAGeek • 4 months ago

for the line "struct node* next;" it asks me to initialize before using. Should i write "struct node* next = *head_ref;" ?

• Reply • Share ›

codecrecker • 6 months ago

#include<stdio.h>

#include<stdlib.h>

typedef struct nd node;

struct nd{

int d;

node *n;

};

node *head,*ptr;

node *createNode(int d)

{

node *tmp;

see more

1 • Reply • Share ›

pratyush • 6 months ago

Is something wrong with *head=*head->next?

• Reply • Share ›

ryan ➚ pratyush • 6 months ago

no just use lyk this *head=(*head)->next

• Reply • Share ›

Guest ➚ ryan • 2 months ago

Can we use *head->next=*head instead?

• Reply • Share ›

ryan Guest • 2 months ago
no
• Reply • Share ›

Jeevan Reddy Mandali • 6 months ago
Hey please someone clear my doubt, if i delete a node and later i try to print the data of that node what should be the output? How to check if the node is deleted or not?
• Reply • Share ›

Hello_world Jeevan Reddy Mandali • 4 months ago
alway fill pointer with NULL after freeing the memory. Even though you have free the memory but variable will still contain the pointer value. Good practice is NULL check before using the memory.

Always initialize pointer with NULL and after freeing it fill it with NULL.

free(node);
node = NULL;
2 • Reply • Share ›

Rocky Jeevan Reddy Mandali • 6 months ago
If you delete a node and then try to print/access it's data, you would lead to a runtime error. To check if the node has been deleted or not, you may check if the node is null or not. If the node is null, the node has been deleted.
1 • Reply • Share ›

Jeevan Reddy Mandali Rocky • 6 months ago
Thx man
• Reply • Share ›

Brajesh Kumar • 7 months ago
A recursive way:

```
/* Function to delete the entire linked list */
void deleteList(struct node** head_ref)
{
/* deref head_ref to get the real head */
if(*head_ref == NULL)
return ;
struct node* temp = *head_ref;
deleteList(&(*head_ref)->next);
printf("\n freeing node : %d", temp->data);
free(temp);
```

*head_ref = NULL;
}

• Reply • Share ›

Learner • 8 months ago

//It's running for infinite times.....what's the problem

#include<iostream>

using namespace std;

struct node

{

int key;

struct node * next;

} ;

struct node * push(struct node * head, int key)

{

struct node * next=new node;

see more

• Reply • Share ›

Gabriel B Nongsiej Learner • 7 months ago

Return <head> from the delete function. It will set the head pointer in the main() to NULL, which makes it possible to display the list without accessing an incorrect address.

• Reply • Share ›

DS+Algo=Placement Learner • 8 months ago

Try to pass double pointer of head to delete() function.

1 • Reply • Share ›

Saurabh • 9 months ago

void deleteList(node **head)

{

if(head==null)

return ; ///list is already empty

```
node * t = *head;

while(*head!=null)

{

*head= t->next;

delete t;

t = *head;

}

}
```

• Reply • Share ›

SANTOSH KUMAR MISHRA • 10 months ago

```
node *DeleteList(node *head)

{

node *prev,*temp,*ptr = head;

if(head == NULL)
{
printf("\nList is empty.");
return head;
}
while(ptr != NULL)
{
temp = ptr;
prev = ptr->next;
ptr = prev;
free(temp);
}
head = ptr;
return head;
}
```

3 • Reply • Share ›

Gaurav Nara • 10 months ago

ONE DOUBT::
Recursive method and the iterative method both have O(n) time complexity, but someone has commented that recursive space complexity will be more !
Anyone who can explain??

Anyone who can explain? ?

• Reply • Share ›

Legolas ➚ Gaurav Nara • 9 months ago

For every recursive function call space is allocated on the stack for local variable,ptr to the return location etc.Also recursive method require more time to make function call negligible though.Hence Iterative method are preferred over the recursive one.If you have a tail recursive call it is optimize by the compiler to the iterative.Fell free to ask more question if you don't get it.

1 • Reply • Share ›

Gaurav Nara ➚ Legolas • 9 months ago

btw Thanks :)

• Reply • Share ›

Gaurav Nara ➚ Legolas • 9 months ago

Yeah even i too have seen this. My iterative method takes less time than the almost same recursive code of it. So is it like if we can find the iterative method then we should go with it instead of doing it with recursion ?

• Reply • Share ›

Legolas ➚ Gaurav Nara • 9 months ago

Yes Iterative methods are definitely preferred over the recursive one but readability and code maintenance are also important as recursion make code easy to understand they are preferred over iterative approach when code become complex.For example Tree traversal using recursion is easy to understand but it become more complex when we try to do it iteratively.

• Reply • Share ›

popeye • 10 months ago

A suggestion: you can use valgrind to easily check whether all the memory has been properly freed.

• Reply • Share ›

Himanshu Dagar • a year ago

http://ideone.com/7RkLNM

can refer to this

• Reply • Share ›

mahesh • a year ago

Recursive solution for the same problem.

```
void deleteList(struct node **head)
```

```
void deleteList(struct node** head)
{
    if(*head)
       {
            deleteList(&(*head)->next);
            free(*head);
            *head = NULL;
        }
}
```

4 • Reply • Share ›

hello mahesh • 8 months ago

sorry to say bt this code of yours work infinite times! could anyone give a suggestion!

• Reply • Share ›

mahesh hello • 8 months ago

Will you please point out any particular case cause infinite times execution of function.

1 • Reply • Share ›

jayasurya j mahesh • a year ago

i have a doubt! can u explain me *head = NULL ... does this mean everytime NULL is assigned to a node?

• Reply • Share ›

Akanksha jayasurya j • 9 months ago

Yes, every time it set to NULL, otherwise your head point to some location(dangling pointer) which is not exist.

• Reply • Share ›

anon • a year ago

I haven't understood the last line in the deleteList function.When I run the function without doing *head_ref = NULL and call a function to find the length of the list it gives the correct length.

So my question is what happens if we do not write the last line in deleteList and why?

• Reply • Share ›

asunel • a year ago

@GeeksforGeeks: Is it possible to delete a linked list having a loop without removing the loop?

• Reply • Share ›

nitin • 2 years ago

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

```
struct node
{
    int data;
    struct node * link;
};
void insert1(struct node **p,int data)
{
    struct node *temp,*t;
    temp=(struct node *)malloc(sizeof(struct node));
    temp->data=data;
    temp->link=NULL;
    if((*p)==NULL)
    {
        *p=temp;
    }
    else
```

see more

2 • Reply • Share ›

Ankit Malhotra • 2 years ago

A simple tail recursive C++ function instead of head recursion examples seen before, that deletes nodes and then passes to next.

```
 void rdellist (node * &ptr) {
  if (!ptr) return;
  node * temp = ptr->next;
  delete ptr;
  rdellist (ptr = temp);
}
```

Recursion is best avoided with loops where feasible so

```

```

void dellist (node * &ptr) {
node * temp;
while (ptr) {
temp = ptr->next;
delete ptr;
ptr = temp;
}
}

• Reply • Share ›

Harsh Agarwal • 2 years ago

A recursive code:

```
 void delete_list(node1 *start)
{
     if(start==NULL)
     {
                  printf("Empty linked list...\n");
                  return;
     }
     node1 *p=start;
     if(p->link==NULL)
     free(p);
     else
     delete_list(p->link);
     p->link=NULL;
}
```

• Reply • Share ›

Ankit Malhotra Harsh Agarwal • 2 years ago

Segmentation Fault. After free(p) reference to p->link is invalid.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

1 • Reply • Share ›

abhishek08aug Ankit Malhotra • 2 years ago

Corrected version :)

```
 /* Function to delete the entire linked list */
void deleteList(struct node** head_ref)
{
  if(*head_ref==NULL) {
    return;
  } else {
    deleteList(&((* head_ref)->next));
    free(*head_ref);
  }
}
```

1 • Reply • Share ›

Chandrashis Mazumdar ➦ abhishek08aug • 7 months ago

at the end *head_ref=NULL; should be there no?

• Reply • Share ›

neelabhsingh ➦ abhishek08aug • 2 years ago

In recursive time complexity will be O(n) but space complexity will be increase from O(1) to O(n).
Abhisek there is some improvement in your code. If you want to display the node it will go infinite. In if condition is only for to reach last node then start deletion. Now you can see on reaching start node you free. But in your function you passed the address of head node double pointer. But you did not make it NULL. If i am wrong plz correct me. Code should be following

```
NODE * deleteList(NODE **start)
{
if(*start==NULL)
{
printf("Now head is NULL");
return NULL;
}
else
{
deleteList(&((*start)->next));
free(*start);
return NULL;
}
}
```

• Reply • Share ›

Sunil • 2 years ago

A recursive algorithm to delete the linked list :)

```
 void delete_list(struct node** node)
{
    if(NULL == *node)
        return;
    delete_list((*node)->link);
    free(*node);
}
```

• Reply • Share ›

Gupta ➦ Sunil • 2 years ago

Wrong code . U just need 2 pass d adress. make it simple

Wrong code.. U just need 2 pass d adress. make it simple..

```
void deleteNode(struct node* head)
{
if(head==NULL)
return;
else
{
delete(head->next);
free(head);
}
}
```

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

neelabhsingh Sunil • 2 years ago

In recursive time complexity will be O(n) but space complexity will be increase.

```
/* Paste your code here (You may delete these lines if not writing code) */
```

• Reply • Share ›

abhishek08aug Sunil • 2 years ago

Wrong code Sunil. you should change

delete_list((*node)->link);

to

delete_list(&((*node)->link))

Check my code above.

• Reply • Share ›

kpnigalye • 3 years ago

[sourcecode language="C++"]

```
/* Delete a linked list */
void DeleteList(struct LinkedListnode*& head_ref)
{
struct LinkedListnode* current = head_ref;
if(head_ref == NULL)
return;
while(current!=NULL)
{
```

```
{
struct LinkedListnode* temp = current;
current = current->next;
cout<<"Node deleted is: "<<temp->data<<endl;
delete temp;
}
head_ref = NULL;
}

/* Recursive way to delete a linked list */
```

see more

• Reply • Share ›

amitp49 • 3 years ago

Recursive solution for the same can be...

```
/* Function to delete the entire linked list */
void deleteList(struct node** head_ref)
{
        if(head_ref==NULL)
                return;
        if((*head_ref)->next)
                deleteList(&(*head_ref)->next);
        free(*head_ref);
        *head_ref = NULL;
}
```

• Reply • Share ›

Yogendra Singh Vimal ➚ amitp49 • 2 years ago

i think @amitp49 that ur code is a Li'L bit wrong...u Should write if(*head_ref==NULL)

instead of,

if(head_ref==NULL) ,

as here double pointer " head_ref " receives the address of a single pointer, So writing only head_ref indicates the address of the single pointer but here our intention is towards the address stored inside single pointer which can be achieved by writing *head_ref.

• Reply • Share ›




- **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