Write a C function to print the middle of a given linked list

Method 1:
Traverse the whole linked list and count the no. of nodes. Now traverse the list again till count/2 and return the node at count/2.

Method 2:
Traverse linked list using two pointers. Move one pointer by one and other pointer by two. When the fast pointer reaches end slow pointer will reach middle of the linked list.

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

```c
/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* Function to get the middle of the linked list*/
void printMiddle(struct node *head)
{
    struct node *slow_ptr = head;
    struct node *fast_ptr = head;

    if (head!=NULL)
    {
        while (fast_ptr != NULL && fast_ptr->next != NULL)
        {
            fast_ptr = fast_ptr->next->next;
            slow_ptr = slow_ptr->next;
        }
        printf("The middle element is [%d]\n\n", slow_ptr->data);
    }
}

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;
}

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

/* Drier program to test above function*/
```

```
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;
    int i;

    for (i=5; i>0; i--)
    {
        push(&head, i);
        printList(head);
        printMiddle(head);
    }

    return 0;
}
```

Output:

```
5->NULL
The middle element is [5]

4->5->NULL
The middle element is [5]

3->4->5->NULL
The middle element is [4]

2->3->4->5->NULL
The middle element is [4]

1->2->3->4->5->NULL
The middle element is [3]
```

Method 3:
Initialize mid element as head and initialize a counter as 0. Traverse the list from head, while traversing increment the counter and change mid to mid->next whenever the counter is odd. So the mid will move only half of the total length of the list.
Thanks to Narendra Kangralkar for suggesting this method.

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

/* Link list node */
struct node
{
    int data;
    struct node* next;
};

/* Function to get the middle of the linked list*/
void printMiddle(struct node *head)
{
    int count = 0;
    struct node *mid = head;
```

```
    while (head != NULL)
    {
        /* update mid, when 'count' is odd number */
        if (count & 1)
            mid = mid->next;

        ++count;
        head = head->next;
    }

    /* if empty list is provided */
    if (mid != NULL)
        printf("The middle element is [%d]\n\n", mid->data);
}


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;
}

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

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

    for (i=5; i>0; i--)
    {
        push(&head, i);
```

```
        printList(head);
        printMiddle(head);
    }

    return 0;
}
```

Output:

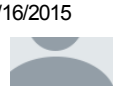
```
5->NULL
The middle element is [5]

4->5->NULL
The middle element is [5]

3->4->5->NULL
The middle element is [4]

2->3->4->5->NULL
The middle element is [4]

1->2->3->4->5->NULL
The middle element is [3]
```

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

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.



// This code prints one as a middle node if the number of elements are odd else it prints 2 elements if the number of elements are even
public void findMiddleElement(Node node)

{

if (node == null)

{

return;

}

else

{

Node nodeToBeIncrementedBy1 = node;

see more

• Reply • Share ›

Vineeth Reddy • 16 days ago

I wouldn't say these are the perfect solutions to solve the question. And when there are even number of inputs, isn't it obvious to print the middle two numbers?

• Reply • Share ›

codebazfam • a month ago

```
void middle()
{
struct node * temp=head;
int c=0;
while(temp!=NULL)
{
temp=temp->next;
c++;
}
int x=c/2;
temp=head;
while(x--)
{
temp=temp->next;
}
cout<<"middle element";
```

cout<<temp->data;
}

• Reply • Share ›

amit sarkar • 3 months ago

method 1 and 3 are same both increment loop by 1.....follow the flow diagram u can understand , method 2 is better since it increment by 2, hence takes n/2 times of loop

1 • Reply • Share ›

Mohammad Zafarul Quadri • 3 months ago

Logically Method 2 and Method 3 are same. In Method 3, mid is behaving as slow pointer and head as fast pointer. But for n nodes; Method 3 loops n times and Method 2 loops n/2 times only. Also, if we have to find middle of middle element i.e., (n/4)th node, then Method 3 may not be used. Am I not correct?

6 • Reply • Share ›

Rushikesh Joshi ➚ Mohammad Zafarul Quadri • 2 months ago

I guess if you want to find (n/4)th node then none of this method works..in method 1 we have to traverse upto length/4 in link list..in method 2 we have to make speed of the faster pointer 4x as compare to slower pointer..in short common logic should be if you want to fetch (n/x)th node from linked list then speed of the faster pointer should be x times more..Please correct me if I am wrong...

5 • Reply • Share ›

Mohammad Zafarul Quadri ➚ Rushikesh Joshi • 2 months ago

Your are right Rushikesh. Your logic is generic solution to such problems!

• Reply • Share ›

RajaCEGian ➚ Mohammad Zafarul Quadri • 3 months ago

Is method two is better as it takes n/2 to find the middle element ?

1 • Reply • Share ›

Adauta Garcia Ariel ➚ RajaCEGian • 2 months ago

So let me se if I understand why is still being O(n), since it will execute n/2 times the 1/2 for asymptotic analysis is just "n", therefore it takes O(n). At first glance I thought it took longer.

• Reply • Share ›

Mohammad Zafarul Quadri ➚ RajaCEGian • 2 months ago

IMO method 2 is better. But time complexity will still be O(n).

• Reply • Share ›

Shobhit Chandrasekaran • 3 months ago

any one bought leetcode ebook? how was it?

any one bought leetcode ebook? how was it?

• Reply • Share ›

ram • 3 months ago

which one is better method 2 or 3 if we see for time complexity

1 • Reply • Share ›

Elangovan Manickam ram • 7 days ago

In my point of view method 2 is better. Because the loop execution is n/2 times, while the method 3 has n times. Though the underlying logic is almost same, we needed a extra variable and calculation in method 3.

• Reply • Share ›

Mohammad Zafarul Quadri ram • 3 months ago

Please see my comment above.

• Reply • Share ›

Amrita • 3 months ago

For Method 3, what if the middle of list is not odd. Like when the list is having even number of nodes.

• Reply • Share ›

ram Amrita • 3 months ago

for 6 nodes it will give ans->4
for 4 nodes it will give 3

• Reply • Share ›

Arun Mittal • 3 months ago

3rd method is better method then 1st i really appreciate u man

1 • Reply • Share ›

davinci • 3 months ago

in method 3 how can we initialise the middle element as head before we even know that it is the middle element.

• Reply • Share ›

tera baap davinci • 2 months ago

tu to bhut hi bakchod hi be

1 • Reply • Share ›

coder.girl • 5 months ago

/* update mid, when 'count' is odd number */
if (count & 1)
Please explain how it gives odd nos?

1 ^ | v • Reply • Share ›

Anurag Singh ➚ coder.girl • 5 months ago

This is bit/binary "and" operation on two numbers.
Think in terms binary representation of count and 1.
1 will look like "00000.....0001", i.e. Only rightmost bit is 1, all other bit zero.
Now if we do binary and of 1 with ANY number, we will get only two results: 1 or 0.
1 will come when rightmost bit is 1 in second number (count)
0 will come if rightmost bit is 0
e.g.
10111001
00000001

00000001

1011001010
0000000001

0000000000

So if "count & 1" is 1, it tells that rightmost bit in count is 1 and so count must be a odd number (any number with rightmost bit 1 will be odd)

6 ^ | v • Reply • Share ›

coder.girl ➚ Anurag Singh • 5 months ago

Thank u so much:)

3 ^ | v • Reply • Share ›

redblood • 5 months ago

what happen in method 2 if the no of node are even ?

^ | v • Reply • Share ›

Anurag Singh ➚ redblood • 5 months ago

It is shown is example output. If number of nodes is N, then ((N/2) + 1)th node is the middle node

^ | v • 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

• Reply • Share ›

suresh • 6 months ago

this code works well.if you want to get the first middle element in even case (1,2,3,4) we get (2)

```
void printMiddle(struct node *head)
{
struct node *slow_ptr = head;
struct node *fast_ptr = head;
if (head!=NULL)
{
while (fast_ptr != NULL && fast_ptr->next != NULL && fast_ptr->next->next != NULL)
{
fast_ptr = fast_ptr->next->next;
slow_ptr = slow_ptr->next;
}
printf("The middle element is [%d]\n\n", slow_ptr->data);
}
}
```

• Reply • Share ›

Sai Madan • 6 months ago

i think this code works well.if you want to get the first middle element in even case (1,2,3,4) we get (2).

```
node *middle(node *head)

{

if(head==NULL||head->link==NULL)
```

return head;

else

{

node *fast=head;

node *middle=head;

while(fast!=NULL)

---

see more

• Reply • Share ›

Sahdev • 6 months ago

complexity wise method 2 is best among all three.
Method 3 is good to impress interviewer. :)

3 • Reply • Share ›

tintin • 6 months ago

In Java

public static int getMiddleElement(LinkedList list) {

Node slow = list.getHead();

Node fast = list.getHead();

while(fast.next != null && fast.next.next != null) {

slow = slow.next;

fast = fast.next.next;

}

return slow.data;

}

• Reply • Share ›

tintin tintin • 6 months ago

For 5 -> 1 -> 2 -> 3 -> 6 -> NULL, middle element is 2

For 5 -> 1 -> 3 -> 6 -> NULL, middle element is 1 and NOT 3 which is the case with the above solutions.

• Reply • Share ›

Neyaz Ahmad • 7 months ago

Method 3 : Java Code

http://ideone.com/2wk53u

• Reply • Share ›

kcolrehs • 7 months ago

method 3 awesome

• Reply • Share ›

Kim Jong-il • 7 months ago

@GeeksforGeeks In method 3: in the while Condition put (head != NULL && head -> next != NULL), then that will work similar to the Method 1 and method 2. Otherthan its giving wrong answer in the case when list have even number of element. Thanks.

• Reply • Share ›

Gut_code ➦ Kim Jong-il • 6 months ago

Method 3

well it does not give wrong answer to me .I think this code will run perfectly, i have run this code on notebook and it works perfectly fine

• Reply • Share ›

Jun • 8 months ago

method 1

http://codepad.org/jPrMWICm

• Reply • Share ›

vikas • 8 months ago

This program works fine until there is 2 nodes in the linklist.

so neither node itself is NULL nor its next pointer is Null, so control enters the loop and increment the fast as well as slow. Whether slow should not be incremented.

so the feasible solution is :

```
while (q->next != NULL)

{

q = q->next;

if (flag)

{
```

```
p = p->next;

}

flag = !flag;

}
```

correct me if im wrong

1 • Reply • Share ›

Guest • 8 months ago

In the 3rd method why did we also change head to head->next?

1 • Reply • Share ›

Rohith Reddy P ➦ Guest • 8 months ago

it is because we are traversing the linked list for maintaining the counter

• Reply • Share ›

Dheeraj Arya • 8 months ago

http://dj8aprilwrites.blogspot...

• Reply • Share ›

Palash • 9 months ago

I think it should be:
while (fast_ptr != NULL && fast_ptr->next->next != NULL)

instead of
while (fast_ptr != NULL && fast_ptr->next != NULL)
in the 2nd method, for the case of even numbers.

2 • Reply • Share ›

Abhinav Mukherjee ➦ Palash • 8 months ago

Its correct, fast_ptr != NULL checks if it passed the last element and fast_ptr->next != NULL checks if it is the last element

1 • Reply • Share ›

Viraj ➦ Abhinav Mukherjee • 8 months ago

I think palash is correct. Can you tell me what will the middle of the list be if there are even number of elements. Say 1->2->3->4. Before the 1st iteration both the slow and fast pointers will be at the head ie 1. After the 1st iteration the slow pointer will be pointing to element 2 and fast pointer pointing to element 3. Shouldn't it stop here ? I might be missing something here.

• Reply • Share ›

Manohar ➚ Viraj • 7 months ago

while (fast_ptr != NULL && fast_ptr->next != NULL) {
if (fast_ptr->next->next == NULL)
break;
fast_ptr = fast_ptr->next->next;
slow_ptr = slow_ptr->next;
}
This code should work.

• Reply • Share ›

Surezh ➚ Viraj • 8 months ago

if the list contains odd number of elements,then fast_ptr.next.next not even exists.So deciding the end of list differs for odd and even number of elements.If the list contains odd number of elements then fast_ptr.next.next has to validated.If the list is of odd numbers then fast_ptr.next has to be validated.
Method 1 is the optimum solution.O(n+n/2)

• Reply • Share ›

Viraj ➚ Surezh • 8 months ago

I think it exists. The condition of the while loop IMO should be while(fast_ptr->next != NULL && fast_ptr->next->next != NULL) Consider this example : 1->2->3->4->5 Now before the 1st iteration slow and fast point to 1. 2nd iteration slow points to 2 fast points to 3. 3rd iteration slow points to 3 and fast points to 5. (fast->next->next of 3). Stop. Now if there are even number of elements. 1->2->3->4. Now before the 1st iteration slow and fast point to 1. 2nd iteration slow points to 2 fast points to 3. Stop.

• Reply • Share ›

Guest ➚ Viraj • 8 months ago

"3rd iteration slow points to 3 and fast points to 5. (fast->next->next of 3). Stop" in the above case only slow_ptr.next.next will be null since Slow points to 3. fast_ptr.next will be pointing to null.

please correct me if i'm wrong

• Reply • Share ›

Viraj ➚ Guest • 8 months ago

"in the above case only slow_ptr.next.next will be null since Slow points to 3. fast_ptr.next will be pointing to null." you mean fast_ptr.next.next. Yes as after the 3rd iteration, fast_ptr.next will be null so the loop will break and the middle element ie 3 will be the answer. Am I missing anything in your question ?

anything in your question ?

• Reply • Share ›

Rutuparna Viraj • 5 months ago

We need to check inside the while loop

```
if(fast_ptr->next->next==NULL)
{
slow_ptr=slow_ptr->next;
break;
}
```

• Reply • Share ›

Sangavi • 9 months ago

till what value should the counter be incremented???

• Reply • Share ›

Programmer • 9 months ago

Method 2 and 3 are same

1 • Reply • Share ›

Load more comments

Subscribe Add Disqus to your site Privacy

-
  - 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

- lebron

  since the array size is 5, it takes constant...

  K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 2 hours ago

- lebron

  merge sort

  K’th Smallest/Largest Element in Unsorted Array | Set 3 (Worst Case Linear Time) · 2 hours ago

- Shubham Sharma

  You saved my time :)

  Searching for Patterns | Set 2 (KMP Algorithm) · 3 hours ago

- Prakhar

  Why so many LOCs, if I'm not wrong (please...

  Largest Sum Contiguous Subarray · 3 hours ago

- Aayush Gupta

  For R4 Q3, Another solution would be to use a...

  Amazon Interview Experience | Set 168 · 4 hours ago

- EigenHarsha

  For Power Of 2, We Simply Doing.. var1 =

  Practo Interview Experience | Set 2 (Off-Campus) · 4 hours ago