Linked List | Set 2 (Inserting a node)

March 13, 2013

We have introduced Linked Lists in the previous post. We also created a simple linked list with 3 nodes and discussed linked list traversal.

In this post, methods to insert a new node in linked list are discussed. A node can be added in three ways
1) At the front of the linked list
2) After a given node.
3) At the end of the linked list.

Add a node at the front: (A 4 steps process)
The new node is always added before the head of the given Linked List. And newly added node becomes the new head of the Linked List. For example if the given Linked List is 10->15->20->25 and we add an item 5 at the front, then the Linked List becomes 5->10->15->20->25. Let us call the function that adds at the front of the list is push(). The push() must receive a pointer to the head pointer, because push must change the head pointer to point to the new node (See this)

Following are the 4 steps to add node at the front.

```
/* Given a reference (pointer to pointer) to the head of a list and an int,
   inserts a new node on the front of the list. */
void push(struct node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct node* new_node = (struct node*) malloc(sizeof(struct node));

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. Make next of new node as head */
```

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

    /* 4. move the head to point to the new node */
    (*head_ref)    = new_node;
}
```

Time complexity of push() is O(1) as it does constant amount of work.

Add a node after a given node: (5 steps process)
We are given pointer to a node, and the new node is inserted after the given node.

```
/* Given a node prev_node, insert a new node after the given prev_node */
void insertAfter(struct node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL)
    {
      printf("the given previous node cannot be NULL");
      return;
    }

    /* 2. allocate new node */
    struct node* new_node =(struct node*) malloc(sizeof(struct node));

    /* 3. put in the data  */
    new_node->data  = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. move the next of prev_node as new_node */
    prev_node->next = new_node;
}
```

Time complexity of insertAfter() is O(1) as it does constant amount of work.

Add a node at the end: (6 steps process)
The new node is always added after the last node of the given Linked List. For example if the given Linked List is 5->10->15->20->25 and we add an item 30 at the end, then the Linked List becomes 5->10->15->20->25->30.
Since a Linked List is typically represented by the head of it, we have to traverse the list till end and then change the next of last node to new node.

Following are the 6 steps to add node at the end.

```
/* Given a reference (pointer to pointer) to the head
   of a list and an int, appends a new node at the end  */
void append(struct node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct node* new_node = (struct node*) malloc(sizeof(struct node));
```

```
    struct node *last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. This new node is going to be the last node, so make next of it as N
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as head */
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}
```

Time complexity of append is O(n) where n is the number of nodes in linked list. Since there is a loop from head to end, the function does O(n) work.

Following is a complete program that uses all of the above methods to create a linked list.

```
// A complete working C program to demonstrate all insertion methods
#include <stdio.h>
#include <stdlib.h>

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

/* Given a reference (pointer to pointer) to the head of a list and an int,
   inserts a new node on the front of the list. */
void push(struct node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct node* new_node = (struct node*) malloc(sizeof(struct node));

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. Make next of new node as head */
    new_node->next = (*head_ref);
```

```
    /* 4. move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Given a node prev_node, insert a new node after the given prev_node */
void insertAfter(struct node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL)
    {
      printf("the given previous node cannot be NULL");
      return;
    }

    /* 2. allocate new node */
    struct node* new_node =(struct node*) malloc(sizeof(struct node));

    /* 3. put in the data  */
    new_node->data  = new_data;

    /* 4. Make next of new node as next of prev_node */
    new_node->next = prev_node->next;

    /* 5. move the next of prev_node as new_node */
    prev_node->next = new_node;
}

/* Given a reference (pointer to pointer) to the head
   of a list and an int, appends a new node at the end  */
void append(struct node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct node* new_node = (struct node*) malloc(sizeof(struct node));

    struct node *last = *head_ref;  /* used in step 5*/

    /* 2. put in the data  */
    new_node->data  = new_data;

    /* 3. This new node is going to be the last node, so make next of it as N
    new_node->next = NULL;

    /* 4. If the Linked List is empty, then make the new node as head */
    if (*head_ref == NULL)
    {
       *head_ref = new_node;
       return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
        last = last->next;
```

```
    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}

// This function prints contents of linked list starting from the given node
void printList(struct node *node)
{
  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* head = NULL;

  // Insert 6.  So linked list becomes 6->NULL
  append(&head, 6);

  // Insert 7 at the beginning. So linked list becomes 7->6->NULL
  push(&head, 7);

  // Insert 1 at the beginning. So linked list becomes 1->7->6->NULL
  push(&head, 1);

  // Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
  append(&head, 4);

  // Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
  insertAfter(head->next, 8);

  printf("\n Created Linked list is: ");
  printList(head);

  getchar();
  return 0;
}
```

Output:

```
Created Linked list is:  1  7  8  6  4
```

You may like to try **Practice MCQ Questions on Linked List**

We will soon be publishing more posts on Linked Lists.

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

Related Questions:

- Binary Heap
- Delete all occurrences of a given key in a linked list
- How to create mergable stack?
- Deque | Set 1 (Introduction and Applications)
- A data structure for n elements and O(1) operations
- Convert left-right representation of a bianry tree to down-right
- Print level order traversal line by line
- C Program for Red Black Tree Insertion

Suyash Soni • a year ago

Why did you use pointer to pointer ? All the operations performed above can also be performed using pointer to node as --

void append(struct node* head_ref, int new_data) instead of void append(struct node** head_ref, int new_data)

please help. Thanks.

17 • Reply • Share ›

Anil Kumar Suyash Soni • 10 months ago

Double pointer is used when you want to change your local_copy of head_pointer. Remember, call-by-value and call-by-reference. It is similar to that

15 • Reply • Share ›

DS+Algo=Placement Anil Kumar • 8 months ago

Are u saying that we can't change the head pointer if we've used the single pointer?

• Reply • Share ›

Forhad Hossain Methun ➚ DS+Algo=Placement • 8 months ago

Rahul JAIN :

yes something like that .
As head is declared locally here ... so inside function we have to send it as call by reference.....

we may do that without pointer to pointer .... by making head pointer Globally ...
(if nedded ..i will give u the code) :)

2 • Reply • Share ›

Chinmay ➚ Forhad Hossain Methun • 7 months ago

What exactly do you mean by making head pointer global?
Thanks.

• Reply • Share ›

Kartik Nagpal ➚ Chinmay • 6 months ago

In Short: When passing by value, such as in above code, make sure you pass pointer to value you want to change. And we want to change value of head pointer, to insert node at beginning and hence as per rule we pass pointer to head pointer.

i'll use a visual representation to help explaining:
Visual -
pointer_to_head -> head -> first_node -> second_node -> .... -> NULL

Now, lets say i want to insert a new_first_node in the beginning, i.e. before first_node.
Visual -
pointer_to_head -> head -> new_first_node -> first_node -> second_node -> .... -> NULL

As you can see now, head is pointing to new_first_node instead first_node.

So, we had to CHANGE value of head pointer such that it points to

see more

7 • Reply • Share ›

Sneha ➚ Kartik Nagpal • 2 months ago

But, we are not using **head_ref anywhere in insert at beg, we are just using *head_ref, so why not pass a only instead of &a, and catch in

using "head_ref, so why not pass a only instead of &a, and catch in *head_ref only? Please explain.

• Reply • Share ›

Kartik Nagpal ➚ Sneha • 2 months ago

You'll need to revisit your basic pointer related concepts. Reading below might help:

1. variable with name "head_ref" is of type struct node**, which means it is a pointer to a pointer to a struct node. Notice "pointer to a pointer".

2. * is a dereference operator, when used in front of a pointer variable, like head_ref, it gives us the value pointed by that pointer.

Example,

head_ref -> head -> node1 .... -> NULL

Notice, head_ref points to variable "head" which itself is a pointer to actual struct node object somewhere on the heap.

3.You asked above "we are just using *head_ref, so why not pass a only instead of &a, and catch in *head_ref only?"

First of all, your q is not worded as per the standard programming

see more

1 • Reply • Share ›

Sneha ➚ Kartik Nagpal • 2 months ago

Thanks..:)

• Reply • Share ›

ANKIT SINGH ➚ Kartik Nagpal • 2 months ago

What if I return the head pointer from the incorrect function.
(Incorrect):
void append(struct node* head, int new_data) {

//insert new node at head
return head;

}

• Reply • Share ›

Kartik Nagpal ➚ ANKIT SINGH • 2 months ago

1. Above code wont compile. The append function signature specifies void return type. But in the function body, you've returned head, which is

of type struct node*. So, type mismatch.

2. If you want to return head pointer from above function :

struct node* append(struct node* head, int new_data) {
//insert new node at head

return head;
}

Then, when you call above function with arguments, you'll have to assign the returned value to head pointer, using the = assignment operator.
Eg.
head = append(head, 4); //Correct
append(head, 4); //Incorrect

2 • Reply • Share ›

ANKIT SINGH ➚ Kartik Nagpal • 2 months ago

Thanks

• Reply • Share ›

Jerry Goyal ➚ Chinmay • 4 months ago

by declaring head outside of main()

• Reply • Share ›

anon ➚ Suyash Soni • 10 months ago

Altough, you can use struct node *head but only when you are sure that list is not empty. Here we are handling the special case of list being empty, so it is required to pass pointer to the pointer to the node as an argument

5 • Reply • Share ›

eightnoteight ➚ anon • 9 months ago

i don't think so, a pointer to a pointer is passed because we copy the pointer data locally(new_Node->next=head). this is just creating a copy of 'head' pointer variable data into new_Node->next, which remains local. so to make the manipulation globally we use pointer to a pointer.
for example, during a swapping we don't change the pointer variable data which makes it local....

void swap(int* p, int* q){
int* r;
r=p;
p=q;
q=r;
}

this is just local, so we don't do that

1 • Reply • Share ›

Jerry Goyal Suyash Soni • 4 months ago

Linked List program with single pointer
#include<stdio.h>
#include <stdlib.h>
struct node{
int data;
struct node* next;
};
struct node* head=NULL;

void insert(int n){
struct node* tmp=(struct node*)malloc(sizeof(struct node));
tmp->data=n;
tmp->next=NULL;
if(head==NULL){
head=tmp;
return;
}
tmp->next=head;

see more

• Reply • Share ›

Rajan Kalra • 3 months ago

How is insertAfter() time complexity O(1), we cannot get access to the previous node without traversing through the list till the previous node. And in that case the worst case can be that we might have to traverse the entire list making the time complexity as O(n). Please correct me if I am wrong, don't want to end up learning wrong cocept..!!

1 • Reply • Share ›

Adauta Garcia Ariel Rajan Kalra • 2 months ago

Yes you a right but at the same time not. Here is my answer.
As you say, traverse the list takes O(n) to find the previous node, but that complexity is for the function that make that work.

insertAfter() represent an algorithm and receive a node, you don't care how you get that node.

Therefore analyzing the complexity of "insertAfter()" just take amount constant of time. O(1)

1 • Reply • Share ›

Nitin Maurya • 8 days ago

how 8 is inserted after node 7 ?

• Reply • Share ›

Kanok • a month ago

can anybody give me a simple full code of first insert in link list...
please help..thanks.

• Reply • Share ›

Sneha • 2 months ago

But, we are not using **head_ref anywhere in insert at beg, we are just using *head_ref, so why not pass a only instead of &a, and catch in *head_ref only? Please explain..

• Reply • Share ›

Guest • 2 months ago

can any one tell what is meaning of

insertAfter(head->next, 8);

means why its inserting after 7.
head have a address of last node which was used in implementing of last node.

• Reply • Share ›

Thiago • 4 months ago

Why in the first example they used pointer to pointer and not in the another one ?

• Reply • Share ›

Adauta Garcia Ariel ➦ Thiago • 2 months ago

Hello, Thiago.

You observation is a really good one.

If you want to modify what head points, you need to pass the memory adress, so you need "pointer to pointer" since head in main() is a pointer.

In push(), please note that the head pointer has to change, this is one answer to your question.

In append() may or may not change, it deppends if the list is empty, if it's empty you need to update the head pointer. (this is other answer to your question)

In insertAfter() the head pointer do not has to change, instead of that you work with the rerences that you receive and you create.

If you do not understant, probably you require more time learning a little more about pointers, It's really complex talk about this issues, but Im pretty sure that we can do it.

2 • Reply • Share ›

dragonwarrior10 • 5 months ago

In the insertAfter function, the pointer prev_node is being passed by value. From what I understand, a local copy of prev_node should be changed. The changes are not reflected in case of append or push if we don't pass the address of the pointer or return the pointer itself in case we pass it by value. Why are the changes being reflected outside the function in this case?

• Reply • Share ›

adithyan • 6 months ago

why struct node** y double star y not single stat

• Reply • Share ›

priya • 6 months ago

hi,
actually for inserting the node at any given what is need to be passed to the function ..in the given prog for inserting at position 3 how it works when we pass the head->next value...how the head is traversed
can u pls explain

• Reply • Share ›

Pranav Kumar Jha • 8 months ago

Is there anything wrong in declaring "head" global?
I didn't have to pass the pointer to each and every function :/
struct node *head = NULL: // outside main

• Reply • Share ›

ANKIT SINGH ➚ Pranav Kumar Jha • 2 months ago

See the given post
http://www.geeksforgeeks.org/h...

• Reply • Share ›

anonymous ➚ Pranav Kumar Jha • 5 months ago

no

• Reply • Share ›

sonu431 • 9 months ago

What does struct node*head=NULL mean? and what if it were like struct node*head; only and we passed it as a pointer. what is the difference b/w these two pointers?

plz answer it anybody with explanation

• Reply • Share ›

CodeMe ➚ sonu431 • 9 months ago

I will breakdown and explain it :

struct node---> a "node" type

struct node * head means a pointer named "head" which will point to "node" type that we have defined.

So now struct node * head=NULL means that the pointer variable head would point to "nothing,which is a known value to the system".It is a good practice to do that.Otherwise it may point any memory in the system,which might not be legal.

Second : Our code behave abnormally if you don't identify in your code.It can use some random value.

Let me know if there is another reason.

2 • Reply • Share ›

Abhishek Pasayat • 9 months ago

suppose we want to insert number 5 after 6 ??here insertafter function just inserts any number after 7 only!!!

thank you!!1

• Reply • Share ›

Heracles • a year ago

Can we not initialize and use an end pointer- to denote the last node, from the very beginning, as and when we construct the linked list, so that we can make the append operation also O(1) ??

• Reply • Share ›

CodeMe ➚ Heracles • 9 months ago

How will you identify the last_node in the beginning ? To identify the last node of the linkedlist ,you need to traverse the list till the end and then update your last_node.It will still take O(n) time.But be careful.If you have added/delete any element at the end then you need to update the last_node .It means this idea will involve some extra work+O(n) time.

Better solution is to traverse each time u want to append at the end.

1 • Reply • Share ›

Hue • a year ago

How about inserting a node at a particular location?
Something like:
void insertAt(struct node* prev_node, int new_data_index)

How to implement it? Same with deletion

• Reply • Share ›

Sushma ➚ Hue • 9 months ago

In linked list there is only one way to identify the current position/node.You need to specify the previous element so that u can trace to the further elements of the array. Linked list doesn't have any random access concept,so you can never specify a particular "position" like you do in array by using index like A[i].

2 • Reply • Share ›

Rachit Chawla • a year ago

Thanks for such a nice discussion. I have a question - What if I have to insert 8 after 6? then what shall we do? inserAfter(___, 8);

• Reply • Share ›

Sushma ➚ Rachit Chawla • 9 months ago

head-next effectively means the "node" having value 7.
So to traverse 1> 7> 8> 6
you should specify like this :
((head>next)>next)>next.Feel free to correct me if i am wrong.

1 • Reply • Share ›

vykunta • a year ago

why (*head ref)=new node

• Reply • Share ›

Himanshu Dagar ➚ vykunta • a year ago

since head_ref contains address of pointer to a node
so to make any node as head ,u have to change the address content at head_ref so that's the above statement tells

(*head_ref)-- value at the head_ref i.e address of head
so it get changed to new node

3 • Reply • Share ›

Adauta Garcia Ariel ➚ vykunta • 3 months ago

head is declared locally in main(), therefore, if you want to modify it passing to a some function and when that function ends the modification still being visible in "head" variable, you need pass it memory address.

let's imagine the following

let's imagine the following

main()
---
struct node* head = NULL;
---

what head contains is NULL.

it address is : (0X24) and you access with &head

When you pass it by reference to push function (ie you pass it memory address 0X24 for example) you do it as follow.

push(&head,3); -> push(struct node* h,data) -> push(0x24,3);

---

see more

• Reply • Share ›

Categories

- Articles (128)
  - Algorithms (24)
  - C (17)
  - C++ (17)
  - Data Structures (29)
  - DBMS (1)
  - Interview Experiences (6)
  - Java (2)
  - Operating Systems (1)
  - Puzzle (15)
  - Searching and Sorting (10)
- Programs (35)
- Quizzes (2,107)
  - Aptitude (2)
  - Computer Science Quizzes (2,104)
    - Algorithms (147)
    - C (208)
    - C++ (129)
    - Computer Organization and Architecture (1)
    - Data Structures (132)
    - DBMS (2)
    - GATE (1,406)
    - Java (51)
    - Operating Systems (28)
  - Web technologies (1)

- # Recent Discussions

  - Zsw-seu

    I agree your answer,it should be n-1

    Bubble Sort · 2 hours ago

  - vedraiyani

    I have run it in codeblock giving output 4

    C | Operators | Question 5 · 9 hours ago

  - dileep kumar Grandhi

    please explain how "s" has come in output

    C | Operators | Question 12 · 11 hours ago

  - ASBGeek

    Can someone explain how the assigement...

    GATE | GATE-CS-2014-(Set-2) | Question 21 · 11 hours ago

  - Zsw-seu

    I think there is a problem with binarySearch....

    Binary Insertion Sort · 20 hours ago

  - Kuch_Bhi

    Please UPDATE the answer. option A is correct.

    GATE | GATE-CS-2003 | Question 57 · 22 hours ago