

[Courses](#)[Login](#)[Suggest an Article](#)

Linked List | Set 2 (Inserting a node)

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

All programs discussed in this post consider following representations of linked list .

C

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

Java

```
// Linked List Class
class LinkedList
{
    Node head; // head of list

    /* Node Class */
```



```
class Node
{
    int data;
    Node next;

    // Constructor to create a new node
    Node(int d) {data = d; next = null; }
}
```

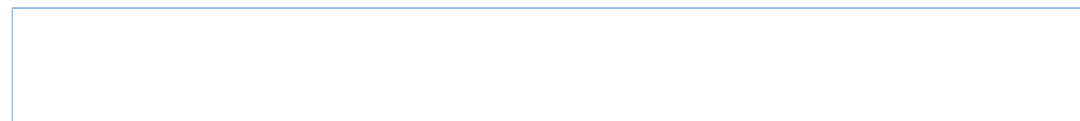
Python

```
# Node class
class Node:

    # Function to initialize the node object
    def __init__(self, data):
        self.data = data # Assign data
        self.next = None # Initialize next as null

# Linked List class
class LinkedList:

    # Function to initialize the Linked List object
    def __init__(self):
        self.head = None
```



In this post, methods to insert a new node in linked list are discussed. A node can be added in three ways

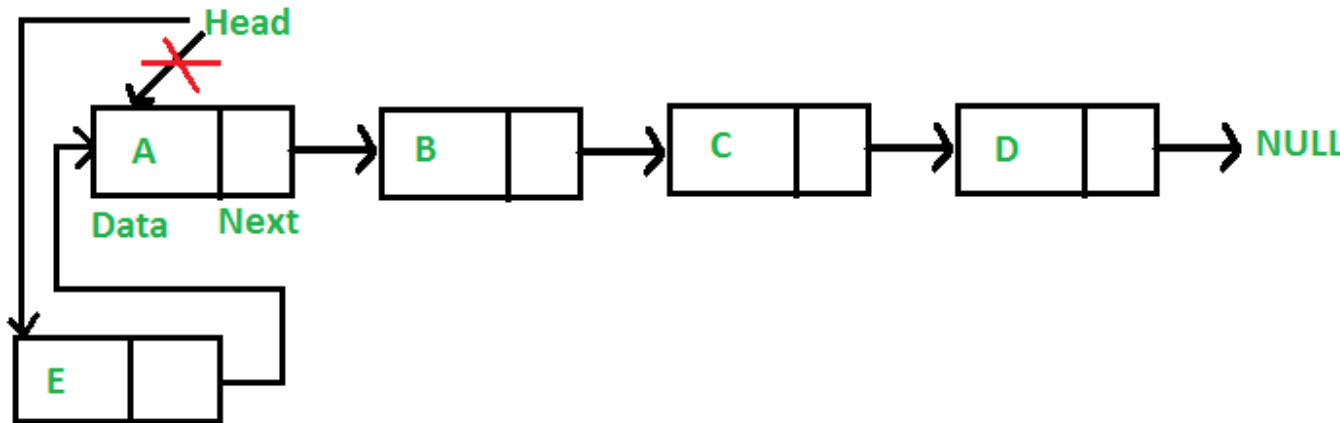
1) At the front of the linked list

- 2) After a given node.
- 3) At the end of the linked list.

Recommended: Please solve it on “PRACTICE” first, before moving on to the solution.

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.

C

```
/* 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;
}
```

Java

```
/* This function is in LinkedList class. Inserts a
   new Node at front of the list. This method is
   defined inside LinkedList class shown above */
public void push(int new_data)
{
    /* 1 & 2: Allocate the Node &
       Put in the data*/
    Node new_node = new Node(new_data);

    /* 3. Make next of new Node as head */
    new_node.next = head;

    /* 4. Move the head to point to new Node */
    head = new_node;
}
```

Python

```
# This function is in LinkedList class
# Function to insert a new node at the beginning
def push(self, new_data):

    # 1 & 2: Allocate the Node &
    # Put in the data
```

```

new_node = Node(new_data)

# 3. Make next of new Node as head
new_node.next = self.head

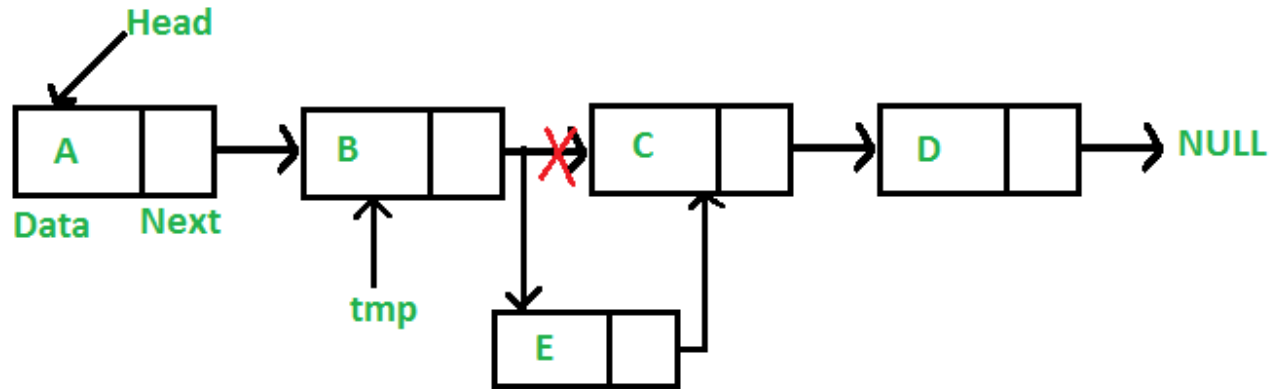
# 4. Move the head to point to new Node
self.head = new_node

```

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.



C

```

/* 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;
}
```

Java

```
/* This function is in LinkedList class.
Inserts a new node after the given prev_node. This method is
defined inside LinkedList class shown above */
public void insertAfter(Node prev_node, int new_data)
{
    /* 1. Check if the given Node is null */
    if (prev_node == null)
    {
        System.out.println("The given previous node cannot be null");
        return;
    }

    /* 2. Allocate the Node &
    3. Put in the data*/
    Node new_node = new Node(new_data);

    /* 4. Make next of new Node as next of prev_node */
    new_node.next = prev_node.next;

    /* 5. make next of prev_node as new_node */
}
```



```
prev_node.next = new_node;
}
```

Python

```
# This function is in LinkedList class.
# Inserts a new node after the given prev_node. This method is
# defined inside LinkedList class shown above */
def insertAfter(self, prev_node, new_data):

    # 1. check if the given prev_node exists
    if prev_node is None:
        print "The given previous node must inLinkedList."
        return

    # 2. Create new node &
    # 3. Put in the data
    new_node = Node(new_data)

    # 4. Make next of new Node as next of prev_node
    new_node.next = prev_node.next

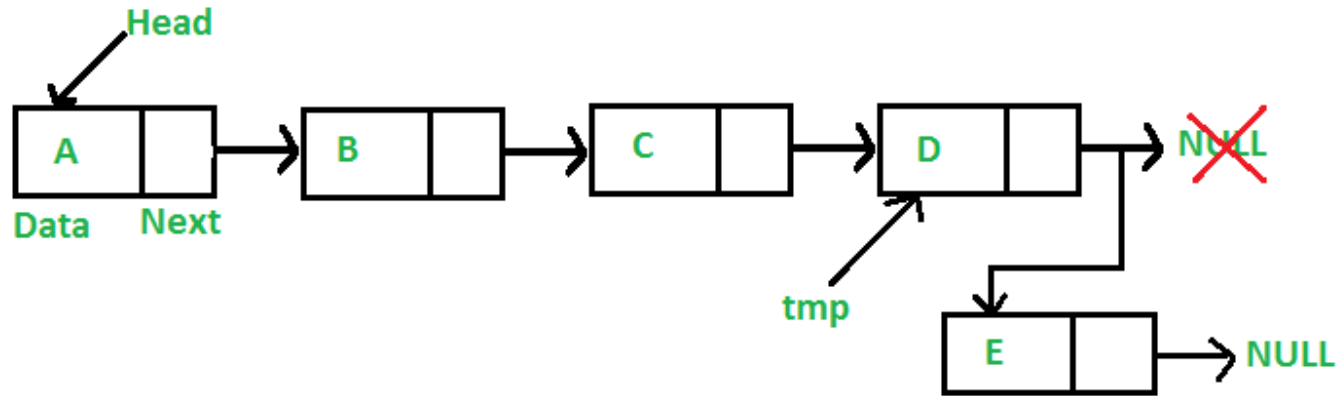
    # 5. make next of prev_node as new_node
    prev_node.next = new_node
```

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.

C

```

/* 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 NULL*/
    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;
}
```

Java

```
/* Appends a new node at the end. This method is
   defined inside LinkedList class shown above */
public void append(int new_data)
{
    /* 1. Allocate the Node &
       2. Put in the data
       3. Set next as null */
    Node new_node = new Node(new_data);

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

    /* 4. This new node is going to be the last node, so
       make next of it as null */
    new_node.next = null;

    /* 5. Else traverse till the last node */
    Node last = head;
    while (last.next != null)
        last = last.next;

    /* 6. Change the next of last node */
}
```



```
    last.next = new_node;
    return;
}
```

Python

```
# This function is defined in Linked List class
# Appends a new node at the end. This method is
# defined inside LinkedList class shown above */
def append(self, new_data):

    # 1. Create a new node
    # 2. Put in the data
    # 3. Set next as None
    new_node = Node(new_data)

    # 4. If the Linked List is empty, then make the
    #     new node as head
    if self.head is None:
        self.head = new_node
        return

    # 5. Else traverse till the last node
    last = self.head
    while (last.next):
        last = last.next

    # 6. Change the next of last node
    last.next = new_node
```

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.

This method can also be optimized to work in $O(1)$ by keeping an extra pointer to tail of linked list/

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

ز ال... احجز ال... احجز ال... احجز ال... احجز ال... احجز ال...

C

```
// A complete working C program to demonstrate all insertion methods
// on Linked List
#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 NULL*/
    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 head
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf(" %d ", node->data);
        node = node->next;
    }
}

/* Driver 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);
}
```



```
    return 0;
}
```

Java

```
// A complete working Java program to demonstrate all insertion methods
// on linked list
```

```
class LinkedList
{
    Node head; // head of list

    /* Linked list Node*/
    class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
           Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Inserts a new node after the given prev_node. */
    public void insertAfter(Node prev_node, int new_data)
    {
        /* 1. Check if the given Node is null */
        if (prev_node == null)
```



```
{
    System.out.println("The given previous node cannot be null");
    return;
}

/* 2 & 3: Allocate the Node &
        Put in the data*/
Node new_node = new Node(new_data);

/* 4. Make next of new Node as next of prev_node */
new_node.next = prev_node.next;

/* 5. make next of prev_node as new_node */
prev_node.next = new_node;
}

/* Appends a new node at the end. This method is
   defined inside LinkedList class shown above */
public void append(int new_data)
{
    /* 1. Allocate the Node &
        2. Put in the data
        3. Set next as null */
    Node new_node = new Node(new_data);

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

    /* 4. This new node is going to be the last node, so
        make next of it as null */
    new_node.next = null;

    /* 5. Else traverse till the last node */
    Node last = head;
    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 */
public void printList()
{
    Node tnode = head;
    while (tnode != null)
    {
        System.out.print(tnode.data+" ");
        tnode = tnode.next;
    }
}

/* Driver program to test above functions. Ideally this function
should be in a separate user class. It is kept here to keep
code compact */
public static void main(String[] args)
{
    /* Start with the empty list */
    LinkedList llist = new LinkedList();

    // Insert 6. So linked list becomes 6->Nulllist
    llist.append(6);

    // Insert 7 at the beginning. So linked list becomes
    // 7->6->Nulllist
    llist.push(7);

    // Insert 1 at the beginning. So linked list becomes
    // 1->7->6->Nulllist
    llist.push(1);

    // Insert 4 at the end. So linked list becomes
    // 1->7->6->4->Nulllist
    llist.append(4);

    // Insert 8, after 7. So linked list becomes
    // 1->7->8->6->4->Nulllist
```




```
l1list.insertAfter(l1list.head.next, 8);

System.out.println("\nCreated Linked list is: ");
l1list.printList();
}
}
// This code is contributed by Rajat Mishra
```

Python

```
# A complete working Python program to demonstrate all
# insertion methods of linked list
```

```
# Node class
```

```
class Node:
```

```
    # Function to initialise the node object
```

```
    def __init__(self, data):
```

```
        self.data = data # Assign data
```

```
        self.next = None # Initialize next as null
```

```
# Linked List class contains a Node object
```

```
class LinkedList:
```

```
    # Function to initialize head
```

```
    def __init__(self):
```

```
        self.head = None
```

```
# Function to insert a new node at the beginning
```

```
def push(self, new_data):
```

```
    # 1 & 2: Allocate the Node &
```

```
    #           Put in the data
```

```
    new_node = Node(new_data)
```

```
    # 3. Make next of new Node as head
```

```
    new_node.next = self.head
```

```
# 4. Move the head to point to new Node
self.head = new_node

# This function is in LinkedList class. Inserts a
# new node after the given prev_node. This method is
# defined inside LinkedList class shown above */
def insertAfter(self, prev_node, new_data):

    # 1. check if the given prev_node exists
    if prev_node is None:
        print "The given previous node must inLinkedList."
        return

    # 2. create new node &
    #    Put in the data
    new_node = Node(new_data)

    # 4. Make next of new Node as next of prev_node
    new_node.next = prev_node.next

    # 5. make next of prev_node as new_node
    prev_node.next = new_node

# This function is defined in Linked List class
# Appends a new node at the end. This method is
# defined inside LinkedList class shown above */
def append(self, new_data):

    # 1. Create a new node
    # 2. Put in the data
    # 3. Set next as None
    new_node = Node(new_data)

    # 4. If the Linked List is empty, then make the
    #    new node as head
    if self.head is None:
        self.head = new_node
        return
```



```
# 5. Else traverse till the last node
last = self.head
while (last.next):
    last = last.next

# 6. Change the next of last node
last.next = new_node

# Utility function to print the linked list
def printList(self):
    temp = self.head
    while (temp):
        print temp.data,
        temp = temp.next

# Code execution starts here
if __name__=='__main__':

    # Start with the empty list
    llist = LinkedList()

    # Insert 6. So linked list becomes 6->None
    llist.append(6)

    # Insert 7 at the beginning. So linked list becomes 7->6->None
    llist.push(7);

    # Insert 1 at the beginning. So linked list becomes 1->7->6->None
    llist.push(1);

    # Insert 4 at the end. So linked list becomes 1->7->6->4->None
    llist.append(4)

    # Insert 8, after 7. So linked list becomes 1 -> 7-> 8-> 6-> 4-> None
    llist.insertAfter(llist.head.next, 8)

    print 'Created linked list is:',
    llist.printList()
```



This code is contributed by Manikantan Narasimhan

Output:

Created Linked list is: 1 7 8 6 4

Linked List | Set 2 (Inserting a node) | GeeksforGeeks



You may like to try [Practice MCQ Questions on Linked List](#)

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



Recommended Posts:

Swap Kth node from beginning with Kth node from end in a Linked List

Linked List | Set 3 (Deleting a node)

Remove last node of the linked list

Remove every k-th node of the linked list

Program for n'th node from the end of a Linked List

Remove first node of the linked list

Squareroot(n)-th node in a Linked List

Sum of nodes in a linked list which are greater than next node

Delete every Kth node from circular linked list

Find the fractional (or n/k - th) node in linked list

Find modular node in a linked list

Write a function to get Nth node in a Linked List

Delete a node in a Doubly Linked List

Insert node into the middle of the linked list

Delete a Linked List node at a given position

الان بسهولة | الان بسهولة | الان بسهولة | الان بسهولة | الان بسهولة | الان بسهولة | الان بسهولة

Article Tags :

Linked List

TCS

Wipro

Practice Tags :

TCS

Wipro

Linked List





47

☐ To-do ☐ Done

1.7

Based on 277 vote(s)

Feedback/ Suggest Improvement

Add Notes

Improve Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Share this post!](#)73 Comments **GeeksforGeeks**[Login](#)[Recommend](#) 2 [Tweet](#) [Share](#)[Sort by Newest](#)

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

**Prem Kumar Megavath** • 6 days ago

What is the difference between `ptr=ptr->next`
And `ptr->next=ptr;`

[^](#) | [v](#) • [Reply](#) • [Share](#)



Tamir Azrab • 7 days ago

Why is it that we are using double pointers? Function in which we adding last node , working fine with just passing head.

```
void addLast ( Node* head , int new_data )
```

While it's not the case in inserting new node at beginning of a list

```
void push ( Node* head , int new_data )
```

It won't work unless head is returned or use of double pointers involved. I mean we're passing pointers then why changing not happening on one end but in other end.

^ | v • Reply • Share ›



Ashish Dwivedi • 12 days ago

why do we check if the given prev_node is NULL

^ | v • Reply • Share ›



TheShivamChand → Ashish Dwivedi • 11 days ago

if the prev_node is NULL, that means there are no nodes, hence the *headRef is not pointing to any node. Now that we know that the list is empty, we can not add the new node. We were supposed to add the element after the previous node.

1 ^ | v • Reply • Share ›



Kody M • a month ago

Using the insertAfter function, how would you insert a node w/ value 8 after the 3rd node w/ value 6?

```
insertAfter(???, 8)
```

I understand, insertAfter(head -> next, 8)

Sorry I am new to programming. Thanks in advance.

^ | v • Reply • Share ›



Tej 007 → Kody M • a month ago

hey kody m,

insertfunction() takes two parameters the first one is address of a node(after which a new node should be inserted) and the second parameter is data of new node. In the function we create a new node and we will add the data into that node and the next part of that node is assigned as value of previous node->next (which you have passed as the first parameter). now we will change the previous node->next to this new node address.

^ | v • Reply • Share ›



Mradul Bohra • 2 months ago

in gcc ubuntu it shows an error that undefined refrencr to traverse list

^ | v • Reply • Share ›



Ashish Shukla • 4 months ago

can any one **help please** ...inserting a node at begining compiler is not giving any error but data is not being displayed...here is the link <https://ide.geeksforgeeks.o...>

^ | v • Reply • Share ›



madan mohan ➔ Ashish Shukla • 24 days ago

Hi Ashish,

In your code, you missed sending the address of the head. Due to that, you won't get output as there won't reference value for head for the function "insert_front".

I made few corrections. Please check it.

```
#include <bits/stdc++.h>
using namespace std;
```

```
class node{
public:
int data;
node* next;
};
```

```
void insert_front(node** head,int n)
{
node* temp=new node;
```

[see more](#)

^ | v • Reply • Share ›



kushal bhatia • 6 months ago

```
new_node->next = prev_node->next;
```


why this next should be pointed to node after new_node
why it is pointing previous node next

^ | v • Reply • Share ›



Alugbin Abiodun → kushal bhatia • 5 months ago

I think this is because the new node will take the space of the previous node. so it is only right to get that value into the new node while the previous node points to the new_node

^ | v • Reply • Share ›



Debajyoti Saha • 7 months ago

why 8 is inserted after 7 instead i think it should add after 1

1 ^ | v • Reply • Share ›



Mukta Malakar → Debajyoti Saha • 7 months ago

The pointer head->next points to the second node. Hence, 8 will come after second node and not after first node.

^ | v • Reply • Share ›



Archito Goswami • 8 months ago

Easy implementation of Linked list insertion and traversal in C at :

<https://marryingpython.blog...>

also see the python of the code at :

<http://marryingpython.blogs...>

^ | v • Reply • Share ›



Saurabh Sharma • 9 months ago

can someone explain me what is actually happening in the 3 step of push().

^ | v • Reply • Share ›



Nivesh Vashist → Saurabh Sharma • 8 months ago

They are simply making the head as the second node of the list. So for that, They created a new node (this will be our new head), and the next field of this head will point the old head.

^ | v • Reply • Share ›



niketanrane → Saurabh Sharma • 9 months ago





I believe you are talking about "new_node->next = (*head_ref);" line. When you are trying to add a newNode at the beginning of a list, your objective is to make the newNode as "HEAD" of the linked list. For that you need to point the newNode's next pointer to current head of the list.

Suppose your list is :

Node1(HEAD_OLD) -> Node2 -> NULL.

The final expected list is :

newNode(HEAD_NEW) -> Node1(HEAD_OLD) -> Node2 -> NULL.

So with the line 3, you are pointing newNode's next to old head of the linked list, similar to :

NewNode ->next point to Node1 (HEAD_OLD)

Which is equivalent to :

new_node->next = (*head_ref)

I hope this helps.

^ | v • Reply • Share ›



divyanshoo sinha • 9 months ago

Hi,

I am trying dynamic insertion of elements in c++ ,but my code is not working can anyone pls help me correct my code below:

```
#include<iostream>
using namespace std;
```

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

```
printlist(Node* n)
{
while(n!=NULL)
{
cout<<n->data;
n=n->next;
}
```



[see more](#) |  • [Reply](#) • [Share](#) ›**rakesh raj** → divyanshoo sinha • 8 months ago


here you have written ,

```
(struct node*)malloc(num*sizeof(struct node));
```

which is point to a dynamic array of size num and contain the value node type **variable**.here are the problems:

1.you already know that in an array ,elements are stored in contiguous fashion so we can determine the address of next element by simply adding one. then why are you taking another pointer variable of type node .

2.we know that size of elements in linked list are not fixed .but here you are created a fixed size(dynamically) blocks.

1  |  • [Reply](#) • [Share](#) ›

**chandan** • 10 months ago

//Dynamic insertion at first,last and any where using Java.

```
import java.util.Scanner;
```

```
class Node{
```

```
int data;
```

```
Node next;
```

```
Node(int data){
```

```
  this.data=data;
```

```
  next=null;
```

```
}
```

```
}
```

```
public class LinkedListInsert {
```

```
  static Node first,second,third;
```

```
  //create list
```

```
  static void createList() {
```

```
    Scanner sc=new Scanner(System.in);
```

```
    System.out.println("enter first node data");
```

```
    int data=sc.nextInt();
```

```
    second=new Node(data);
```

[see more](#) |  • [Reply](#) • [Share](#) ›**divyanshoo sinha** → chandan • 9 months ago



Can you share a c++ code related to the dynamic insertion?

^ | v • Reply • Share ›



Vaibhav Gupta • 10 months ago

Hi,

will using the head pointer as a global variable by declaring it above all the functions, instead of passing the pointer to pointer as a parameter, be a good idea

^ | v • Reply • Share ›



rakesh raj → Vaibhav Gupta • 8 months ago

yes.you can do but it is not a better idea.bcz in

it has many problems like following:

- a) head is globally accessible, so it can be modified anywhere in your project and may lead to unpredictable results.
- b) If there are multiple linked lists, then multiple global head pointers with different names are needed.

See this to know all reasons why should we avoid global variables in our projects.

1 ^ | v • Reply • Share ›



Vaibhav Gupta → rakesh raj • 8 months ago

Yes , right

1 ^ | v • Reply • Share ›



SURYA KUMAR • a year ago

Hi ,

Can some one explain me , how inserting a node after a particular node will result in $O(1)$ time.

It is explained , we already have reference of node after which , the given node is to be inserted.But in actual scenario , we will be given node value right.

So for traversing till that value ,I think even this also will result in $O(n)$

1 ^ | v • Reply • Share ›



Abhay Yadav • a year ago

For the ones who are confused about the pointer to pointer (`**head_ref`), it is being used to change the content inside the head pointer (address of the first node).

If you would use `*head_ref` instead of `**head_ref` and pass head, you will successfully change the list but the head pointer will still point to the same node as it was before passing. This is not good in case if you are adding a first node or a node at the starting.

2 ^ | v • Reply • Share ›



Vaibhav Gupta → Abhay Yadav • 10 months ago

will using head as a global variable be a good idea , so that it can be modified anywhere in the function without passing

^ | v • Reply • Share ›



Nishant Chandel • a year ago

Why this not working????

```
void Insert(struct Node *head, int data)
{
    struct Node *new_Node = (struct Node *)malloc(sizeof(struct Node));
    new_Node->key = data;
    new_Node->next= head;
    head = new_Node;
}
```

^ | v • Reply • Share ›



Vaibhav Gupta → Nishant Chandel • 10 months ago

when we will use pointer instead of double pointer in the insert function and pass the head to it , instead of the real head being taken it will create a copy of the head, although you will be able to do the other operations like traversing to the last node well but changing the content of the head will change the content in that newly created head instead of the original head that we passed, that is why your any attempt to change the head in the function will not give you the desired result.

Understand it like the passing by value funtion , when passing is done by value the original variable is not modified but value of its copy is modified, to modify the original variable we have to pass the pointer to that variable, the below video explains this point well -

<https://www.youtube.com/wat...>

^ | v • Reply • Share ›



Nostradamous → Nishant Chandel • 10 months ago

use:

```
void Insert(struct Node **head,int data)
*head=new_Node
```

Hope it works

1 ^ | v • Reply • Share ›



Nishant Chandel → Nostradamous • 10 months ago



I know this will work, but what is the technical error in that one?

^ | v • Reply • Share ›



Prab • a year ago

Why does the 'append' not work if i use struct Node new_Node and add a '&' appropriately? What is the difference between actually using a pointer and malloc and creating a struct, initializing it, and using it as like:

last -> next = &new_Node

^ | v • Reply • Share ›



aditya • a year ago

Will this append code decrease the time complexity of code ??

```
public void last(int value){
// Node1 new_node = new Node1(value);
// new_node.next = null;
// if(head == null){
// head = new Node1(value);
// return;
// }
//
// Node1 a = head;
// while( a.next !=null){
// a = a.next;
// }
// a.next = new_node;
// return;
Node1 a = head;
Node1 new_node = new Node1(value);
new_node.next = null;

while(a.next != null){
a = a.next;
}
a.next = new_node;
}
```

^ | v • Reply • Share ›





Nagaraju Venkatesh • a year ago

// A complete working Java program to demonstrate all insertion methods

// on linked list

```
class LinkedList
```

```
{
```

```
Node head; // head of list
```

```
/* Linked list Node*/
```

```
class Node
```

```
{
```

```
int data;
```

```
Node next;
```

```
Node(int d) {data = d; next = null; }
```

```
}
```

what is Node head? is Node is a datatype of headpointer?

^ | v • Reply • Share ›



Ashish Kumar Singh → Nagaraju Venkatesh • a year ago

Node head is the object of class node which will store head of linked list.

^ | v • Reply • Share ›



Birajit Nath • a year ago

can any one tell me what is wrong with this code??

```
#include<conio.h>
```

```
#include<iostream>
```

```
using namespace std;
```

```
struct node
```

```
{
```

```
int data;
```

```
node* next;
```

```
};
```

```
node* head;
```

```
void insert(int x)
```



```
{
node* temp=new node();
temp->data=x;
temp->next=NULL;
if(head==NULL)
{
```

[see more](#)

^ | v • Reply • Share ›



Mitesh • a year ago

Why to use **head_ref ?
can i use * head_ref as in function?

^ | v • Reply • Share ›



Udayasankar Thirugnanam • a year ago

Hi,
i have a doubt on function append(..).In 3rd step constructor assigns the value "null" for `new_node.next`, then why again the "null" value assignment repeated at step 4?

```
' public void append(int new_data)
{
/* 1. Allocate the Node &
2. Put in the data
3. Set next as null */
Node new_node = new Node(new_data);

/* 4. If the Linked List is empty, then make the
new node as head */
if (head == null)
{
head = new Node(new_data);
return;
}
```

[see more](#)

^ | v • Reply • Share ›



Bilal Ennouali • a year ago

I don't understand why we should use a pointer of a pointer in the two functions push and append, wouldn't a pointer *head sent to the function instead of the address be enough? Ps: using a pointer instead a pointer of a pointer doesn't work but I don't see why.

1 ^ | v • Reply • Share ›



Eric Henderson • a year ago

Our University teaches everything in c++ now. Would be helpful to see how things change code wise for c++ in this entire series. Any offers to rewrite?

^ | v • Reply • Share ›



Bhart Mittal • a year ago

When prev_node will be NULL in insertAfter function as we are going to pass at least head in calling function in main? (for eg insertAfter(head->next, 8); has inserted 8 after 7 above and insertAfter(head, 8); will going to insert it after first element 1. So there can be no case when prev_node will be NULL)

^ | v • Reply • Share ›



Susanoo • a year ago

In the append function instead of again creating and reference, we can use already existing object to reference.

```
if (head == null)
{
    head = new_node;
    return;
}
```

^ | v • Reply • Share ›



Pratiyush • 2 years ago

Inside public void insertAfter(Node prev_node, int new_data), a check should be done if prev_node exists. What if I add after a node that does not exists in the list?

^ | v • Reply • Share ›



Gowtham48 • 2 years ago

Guys what is wrong with this code

class node:

```
def __init__(self,data):
```



```
self.data=data
self.next= None

class LinkedList:
def __init__(self):
self.head=None

def Print(self):
temp=self.head
while(temp):
print(str(temp.data)+"-> ",end="")
temp=temp.next
print()

def push(self,data):
newnode=node(data)
```

[see more](#)

^ | v • Reply • Share ›



Vipul Bansal • 2 years ago

I don't understand how after "insertAfter(head->next, 8)" 8 is inserted after 7. Can someone please explain ?

1 ^ | v • Reply • Share ›



S Modassir Danish ➔ Vipul Bansal • a year ago

"insertAfter(head->next, 8)" inserts element only after two elements in created list what ever created list.. mostly. It is confusing.

^ | v • Reply • Share ›



Chirayu Jain ➔ Vipul Bansal • a year ago

ya same problem i am facing that why it is inserted after 7 and if head->next is 7 that how i will insert the number at any other place

^ | v • Reply • Share ›

[Show more replies](#)



Harish Govindarajan • 2 years ago

While adding the node at the end, every time when "last = last.next" is done, the previous elements are lost. Is there any solution to it?

 ^ | v • Reply • Share ›**ferb** • 2 years ago



after all the elements are pushed than after that traverse of of link list by passing head as argument which itself is pointing to last element how it is possible

^ | v • Reply • Share ›

**Divyanshi Mangal** ➔ ferb • a year ago

We are creating a dummy node 'last' beginning from head. Then we are traversing until we reach NULL node. It is not pointing to last element in the beginning.

^ | v • Reply • Share ›

[Load more comments](#) [Subscribe](#)  [Add Disqus to your site](#) [Add Disqus](#) [Add Disqus](#)  [Disqus' Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#) [Privacy Policy](#)

A computer science portal for geeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org

COMPANY

About Us
Careers
Privacy Policy
Contact Us

PRACTICE

Company-wise
Topic-wise
Contests
Subjective Questions

LEARN

Algorithms
Data Structures
Languages
CS Subjects
Video Tutorials

CONTRIBUTE

Write an Article
Write Interview Experience
Internships
Videos

@geeksforgeeks, Some rights reserved

