GeeksforGeeks
A computer science portal for geeks

Custom Search

COURSES

HIRE WITH US

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
class Node
{
    public:
    int data;
    Node *next;
};
// This code is contributed by rathbhupendra
```

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
```

C#

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

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

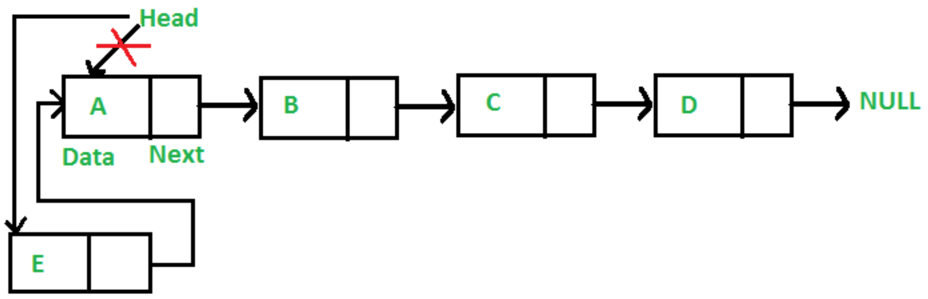
2) After a given node.

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

```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

```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
```

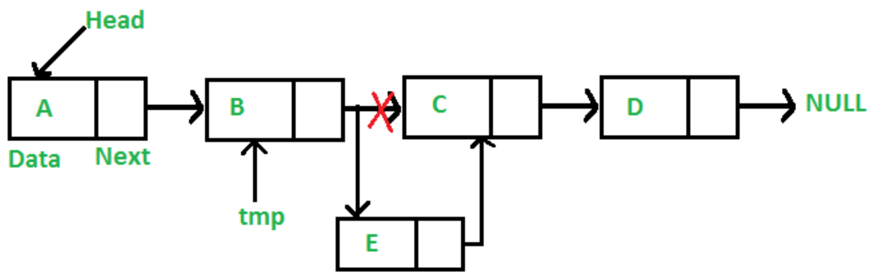
C#

```csharp
/* 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;
}
```

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
```

C#

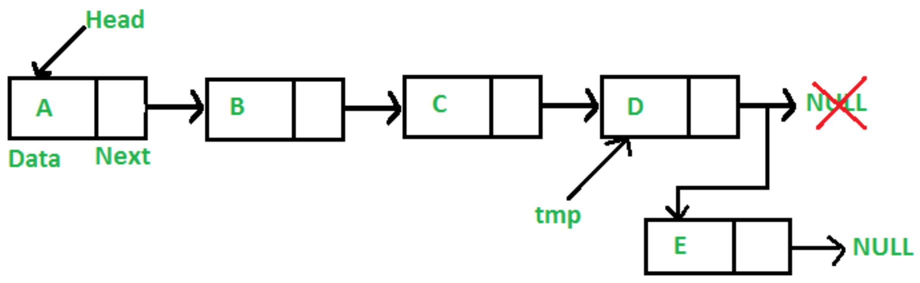
```
/* Inserts a new node after the given prev_node. */
public void insertAfter(Node prev_node,
                        int new_data)
{
    /* 1. Check if the given Node is null */
    if (prev_node == null)
    {
        Console.WriteLine("The given previous node" +
                                " 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;
}
```

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
```

C#

```
/* 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;
}
```

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.

C++

```
// A complete working C++ program to demonstrate
//  all insertion methods on Linked List
#include <bits/stdc++.h>
using namespace std;

// A linked list node
class Node
{
    public:
    int data;
    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(Node** head_ref, int new_data)
{
    /* 1. allocate node */
    Node* new_node = new 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(Node* prev_node, int new_data)
```

```
{
    /*1. check if the given prev_node is NULL */
    if (prev_node == NULL)
    {
        cout<<"the given previous node cannot be NULL";
        return;
    }

    /* 2. allocate new node */
    Node* new_node = new 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(Node** head_ref, int new_data)
{
    /* 1. allocate node */
    Node* new_node = new Node();

    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(Node *node)
{
    while (node != NULL)
    {
        cout<<" "<<node->data;
        node = node->next;
    }
}

/* Driver code*/
int main()
{
    /* Start with the empty list */
    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);

    cout<<"Created Linked list is: ";
    printList(head);

    return 0;
}


// This code is contributed by rathbhupendra
```

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->NUllist
        llist.append(6);

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

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

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

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

        System.out.println("\nCreated Linked list is: ");
        llist.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


    # Functio 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
```

C#

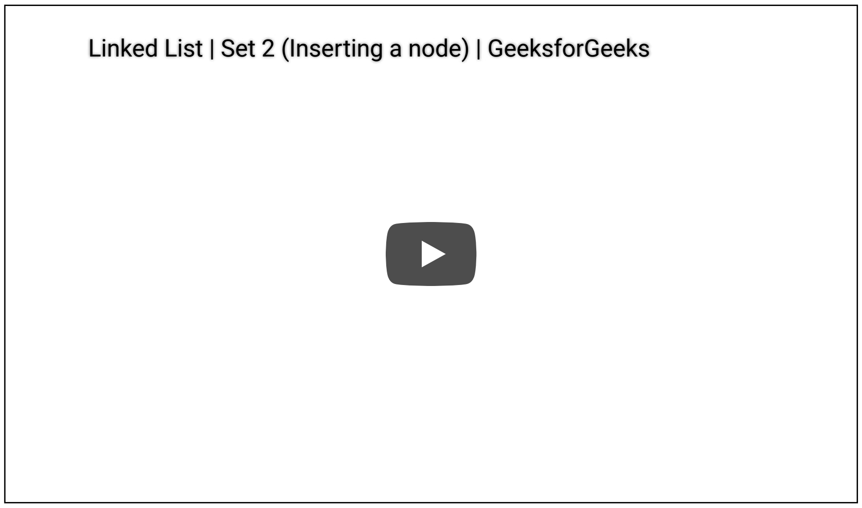
```
// A complete working C# program to demonstrate
// all insertion methods on linked list
using System;
```

```
class GFG
{
public Node head; // head of list

/* Linked list Node*/
public class Node
{
public int data;
public Node next;
public 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)
{
Console.WriteLine("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)
{
Console.Write(tnode.data + ” “);
tnode = tnode.next;
}
}

// Driver Code
public static void Main(String[] args)
{
/* Start with the empty list */
GFG llist = new GFG();

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

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

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

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

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

Console.Write(“Created Linked list is: “);
llist.printList();
}
}
```

// This code is contributed by Rajput-Ji

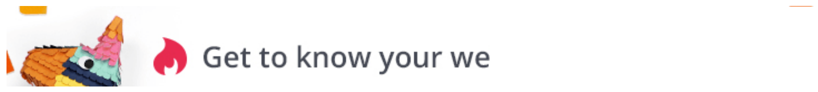
Output:

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



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:

Create new linked list from two given linked list with greater element at each node
Swap Kth node from beginning with Kth node from end in a Linked List
Squareroot(n)-th node in a Linked List
Linked List | Set 3 (Deleting a node)
Delete Nth node from the end of the given linked list
Remove first node of the linked list
Remove last node of the linked list
Program for n'th node from the end of a Linked List
Remove every k-th node of the linked list
Remove Nth node from end of the Linked List
Find modular node in a linked list
Reverse each word in a linked list node
Find the fractional (or n/k - th) node in linked list
Insert node into the middle of the linked list
Find the balanced node in a Linked List

Improved By : rathbhupendra, Rajput-Ji

Article Tags : Linked List TCS Wipro

Practice Tags : TCS Wipro Linked List

93

☐ To-do ☐ Done

1.7

Based on **305** vote(s)

Feedback/ Suggest Improvement 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.

Important Update

When you log in with Disqus, we process personal data to facilitate your authentication and posting of comments. We also store the comments you post and those comments are immediately viewable and searchable by anyone around the world.

Please access our Privacy Policy to learn what personal data Disqus collects and your choices about how it is used. All users of our service are also subject to our Terms of Service.

Proceed

WORLDQUANT UNIVERSITY
We're training the world's next data scientists.... for free.
AP
SEPT