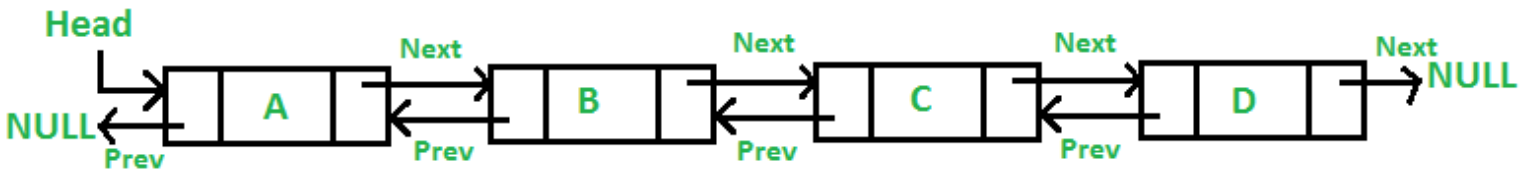
Doubly Linked List | Set 1 (Introduction and Insertion)

We strongly recommend to refer following post as a prerequisite of this post.

Linked List Introduction

Inserting a node in Singly Linked List

A **D**oubly **L**inked **L**ist (DLL) contains an extra pointer, typically called *previous pointer*, together with next pointer and data which are there in singly linked list.



Following is representation of a DLL node in C language.

```
/* Node of a doubly linked list */
struct node
{
  int data;
  struct node *next; // Pointer to next node in DLL
  struct node *prev; // Pointer to previous node in DLL
};
```

Run on IDE

Following are advantages/disadvantages of doubly linked list over singly linked list.

Advantages over singly linked list

1) A DLL can be traversed in both forward and backward direction.

2) The delete operation in DLL is more efficient if pointer to the node to be deleted is given.

In singly linked list, to delete a node, pointer to the previous node is needed. To get this previous node, sometimes the list is traversed. In DLL, we can get the previous node using previous pointer.

Disadvantages over singly linked list

1) Every node of DLL Require extra space for an previous pointer. It is possible to implement DLL with single pointer though (See this and this).

2) All operations require an extra pointer previous to be maintained. For example, in insertion, we need to modify

previous pointers together with next pointers. For example in following functions for insertions at different positions, we need 1 or 2 extra steps to set previous pointer.

Insertion

A node can be added in four ways

1) At the front of the DLL

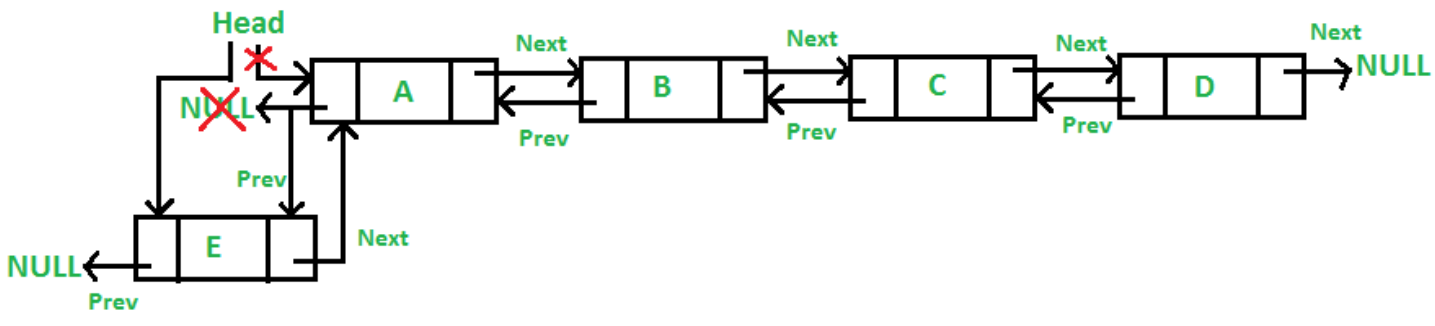
2) After a given node.

3) At the end of the DLL

4) Before a given node.

1) Add a node at the front: (A 5 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 DLL. 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 5 steps to add node at the front.

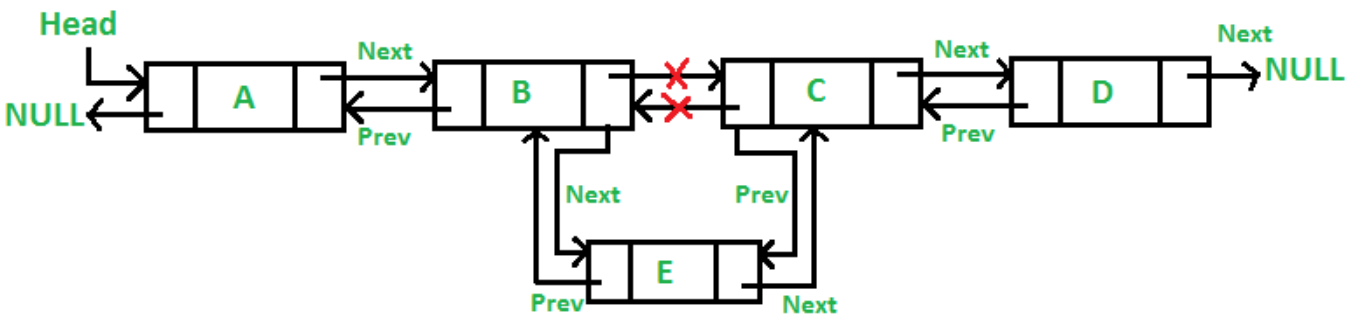
```
/* Given a reference (pointer to pointer) to the head of a list
   and an int, inserts a new node on the front of the list. */
void push(struct node** head_ref, int new_data)
{
    /* 1. allocate node */
    struct node* new_node = (struct node*) malloc(sizeof(struct node));

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

    /* 3. Make next of new node as head and previous as NULL */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if((*head_ref) !=  NULL)
      (*head_ref)->prev = new_node ;

    /* 5. move the head to point to the new node */
    (*head_ref)    = new_node;
}
```

Run on IDE

Four steps of the above five steps are same as the 4 steps used for inserting at the front in singly linked list. The only extra step is to change previous of head.

2) Add a node after a given node.: (A 7 steps process)

We are given pointer to a node as prev_node, and the new node is inserted after the given node.




```
/* Given a node as prev_node, insert a new node after the given 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. Make the next of prev_node as new_node */
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
      new_node->next->prev = new_node;
}
```

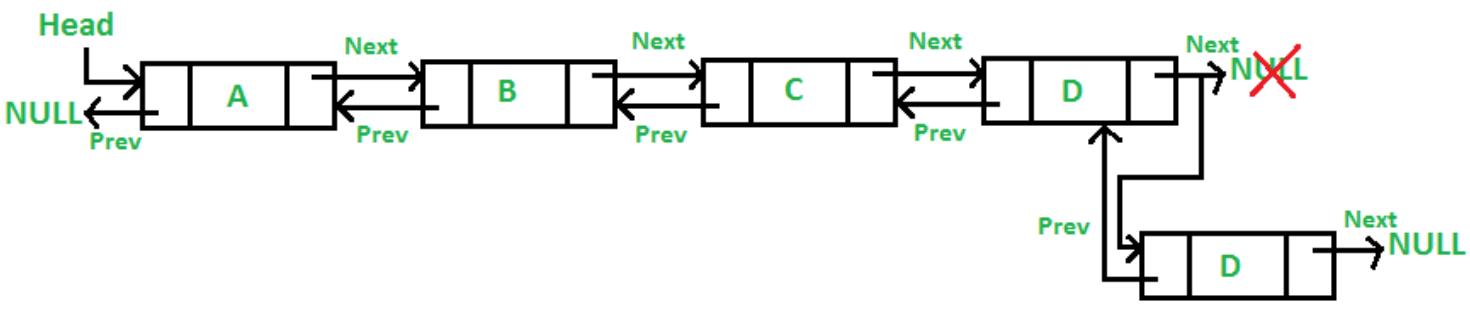
Run on IDE

Five of the above steps step process are same as the 5 steps used for inserting after a given node in singly linked list. The two extra steps are needed to change previous pointer of new node and previous pointer of new node's next node.

3) Add a node at the end: (7 steps process)

The new node is always added after the last node of the given Linked List. For example if the given DLL is 5<->10<-

>15<->20<->25 and we add an item 30 at the end, then the DLL 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 7 steps to add node at the end.

```
/* Given a reference (pointer to pointer) to the head
   of a DLL 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)
    {
        new_node->prev = 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;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}
```

Run on IDE

Six of the above 7 steps are same as the 6 steps used for inserting after a given node in singly linked list. The one extra step is needed to change previous pointer of new node.

4) Add a node before a given node

This is left as an exercise for the readers.

A complete working program to test above functions.

Following is complete C program to test above functions.

```c
// 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;
    struct node *prev;
};

/* 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 and previous as NULL */
    new_node->next = (*head_ref);
    new_node->prev = NULL;

    /* 4. change prev of head node to new node */
    if((*head_ref) !=  NULL)
      (*head_ref)->prev = new_node ;

    /* 5. move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Given a node as prev_node, insert a new node after the given 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. Make the next of prev_node as new_node */
```

```
    prev_node->next = new_node;

    /* 6. Make prev_node as previous of new_node */
    new_node->prev = prev_node;

    /* 7. Change previous of new_node's next node */
    if (new_node->next != NULL)
      new_node->next->prev = new_node;
}

/* Given a reference (pointer to pointer) to the head
   of a DLL 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)
    {
        new_node->prev = 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;

    /* 7. Make last node as previous of new node */
    new_node->prev = last;

    return;
}

// This function prints contents of linked list starting from the given node
void printList(struct node *node)
{
    struct node *last;
    printf("\nTraversal in forward direction \n");
    while (node != NULL)
    {
        printf(" %d ", node->data);
        last = node;
        node = node->next;
    }

    printf("\nTraversal in reverse direction \n");
    while (last != NULL)
    {
        printf(" %d ", last->data);
        last = last->prev;
```

```
    }
}

/* 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("Created DLL is: ");
    printList(head);

    getchar();
    return 0;
}
```

Run on IDE

Python

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

# A linked list node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.next = None
        self.prev = None

# Class to create a Doubly Linked List
class DoublyLinkedList:

    # Constructor for empty Doubly Linked List
    def __init__(self):
        self.head = None

    # Given a reference to the head of a list and an
    # integer,inserts a new node on the front of list
    def push(self, new_data):

        # 1. Allocates node
        # 2. Put the data in it
        new_node = Node(new_data)
```

```
        # 3. Make next of new node as head and
        # previous as None (already None)
        new_node.next = self.head

        # 4. change prev of head node to new_node
        if self.head is not None:
            self.head.prev = new_node

        # 5. move the head to point to the new node
        self.head = new_node

    # Given a node as prev_node, insert a new node after
    # the given node
    def insertAfter(self, prev_node, new_data):

        # 1. Check if the given prev_node is None
        if prev_node is None:
            print "the given previous node cannot be NULL"
            return

        # 2. allocate new node
        # 3. put in the data
        new_node = Node(new_data)

        # 4. Make net of new node as next of prev node
        new_node.next = prev_node.next

        # 5. Make prev_node as previous of new_node
        prev_node.next = new_node

        # 6. Make prev_node ass previous of new_node
        new_node.prev = prev_node

        # 7. Change previous of new_nodes's next node
        if new_node.next is not None:
            new_node.next.prev = new_node

    # Given a reference to the head of DLL and integer,
    # appends a new node at the end
    def append(self, new_data):

        # 1. Allocates node
        # 2. Put in the data
        new_node = Node(new_data)

        # 3. This new node is going to be the last node,
        # so make next of it as None
        new_node.next = None

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

        # 5. Else traverse till the last node
        last = self.head
        while(last.next is not None):
            last = last.next

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

        # 7. Make last node as previous of new node
```

```
        new_node.prev = last

        return

    # This function prints contents of linked list
    # starting from the given node
    def printList(self, node):

        print "\nTraversal in forward direction"
        while(node is not None):
            print " %d" %(node.data),
            last = node
            node = node.next

        print "\nTraversal in reverse direction"
        while(last is not None):
            print " %d" %(last.data),
            last = last.prev

# Driver program to test above functions

# Start with empty list
llist = DoublyLinkedList()

# Insert 6. So the 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 DLL is: ",
llist.printList(llist.head)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Run on IDE

Output:

```
 Created DLL is:
Traversal in forward direction
 1  7  8  6  4
Traversal in reverse direction
 4  6  8  7  1
```

Also see – Delete a node in double Link List

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

GATE CS Notes (According to Official GATE 2017 Syllabus)

GATE CS Corner

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.

Category: Linked List

(Login to Rate and Mark)

Average Difficulty : **1.6/5.0**

1.6

Based on **55** vote(s)

Average Rating : **5/5.0**

Based on **1** vote(s)

★★★★★

Add to TODO List

Mark as DONE

Load Comments