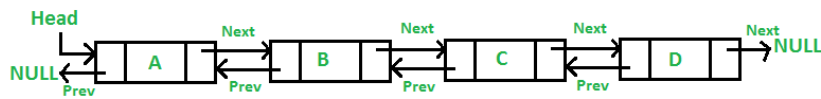# Doubly Linked List in C++

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 */
class Node
{
    public:
    int data;
    Node* next; // Pointer to next node in DLL
    Node* prev; // Pointer to previous node in DLL
};
```

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.

3) We can quickly insert a new node before a given node.

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 10152025 and we add an item 5 at the front, then the Linked List becomes 510152025. 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)
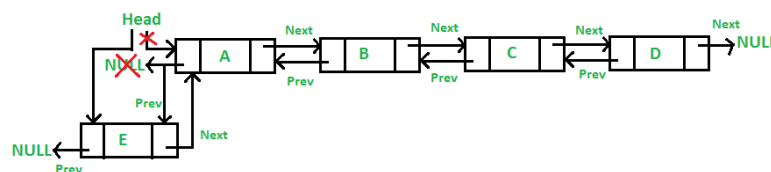
Following are the 5 steps to add node at the front.

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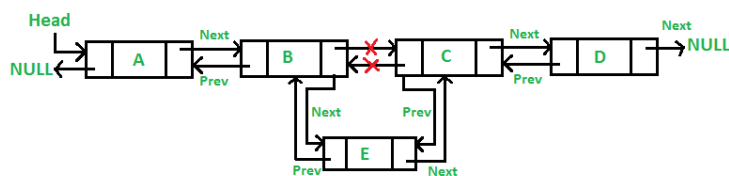
Four steps of the above five steps are same as the 4 steps used for inserting at the front in singly linked list. The only extra step is to change previous of head.

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(Node* prev_node, int new_data)
{
    /*1. check if the given prev_node is NULL */
```

```cpp
    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. 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;
}
```

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 510152025 and we add an item 30 at the end, then the DLL becomes 51015202530.

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.
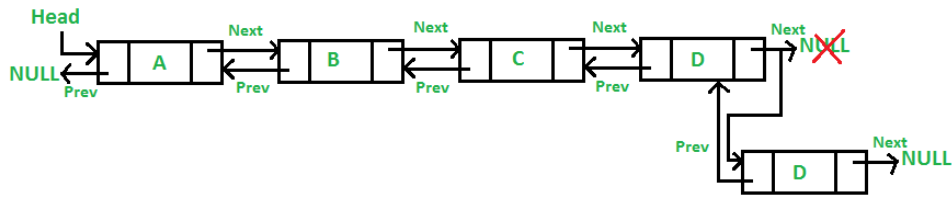
Following are the 7 steps to add node at the end.:

```cpp
/* Given a reference (pointer to pointer) to the head
of a DLL 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)
    {
        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;
}
```

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:

Steps

Let the pointer to this given node be next_node and the data of the new node to be added as new_data.

1. Check if the next_node is NULL or not. If it's NULL, return from the function because any new node can not be added before a NULL

2. Allocate memory for the new node, let it be called new_node

3. Set new_node->data = new_data

4. Set the previous pointer of this new_node as the previous node of the next_node, new_node->prev = next_node->prev

5. Set the previous pointer of the next_node as the new_node, next_node->prev = new_node

6. Set the next pointer of this new_node as the next_node, new_node->next = next_node;

7. If the previous node of the new_node is not NULL, then set the next pointer of this previous node as new_node, new_node->prev->next = new_node

8. Else, if the prev of new_node is NULL, it will be the new head node. So, make (*head_ref) = new_node. Below is the implementation of the above approach:
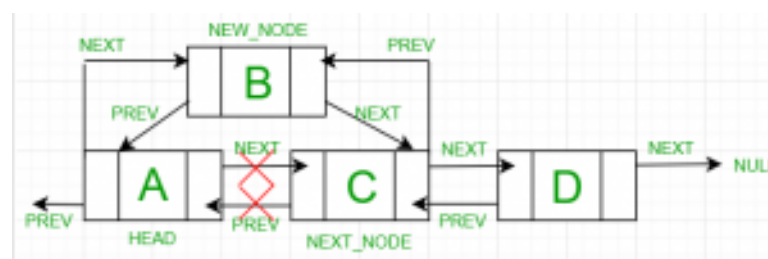
Code block

Output:

Created DLL is:

Traversal in forward Direction

9 1 5 7 6

Traversal in reverse direction

6 7 5 1 9

A complete working program to test above functions.

Following is complete program to test above functions.

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

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

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

```cpp
    /* 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(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. 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;
```

```cpp
    /* 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(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)
    {
        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;
```

```cpp
    /* 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(Node* node)
{
    Node* last;
    cout<<"\nTraversal in forward direction \n";
    while (node != NULL)
    {
        cout<<" "<<node->data<<" ";
        last = node;
        node = node->next;
    }

    cout<<"\nTraversal in reverse direction \n";
    while (last != NULL)
    {
        cout<<" "<<last->data<<" ";
        last = last->prev;
    }
}

/* Driver program to test above functions*/
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
```

```cpp
    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 DLL is: ";
    printList(head);

    return 0;
}
```

Output:

```
Created DLL is:
Traversal in forward Direction
1 7 5 8 6 4
Traversal in reverse direction
4 6 8 5 7 1
```