Lab 3: Doubly Linked List

CLO:
02

Objectives:
To write a program that implements the basic operations of a doubly linked list.

Doubly Linked Lists (DLL) Review
Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List. Following are the important terms to understand the concept of doubly linked list.
Link − Each link of a linked list can store a data called an element.
Next − Each link of a linked list contains a link to the next link called Next.
Prev − Each link of a linked list contains a link to the previous link called Prev.
LinkedList − A Linked List contains the connection link to the first link called First and to the last link called Last.

Representation:

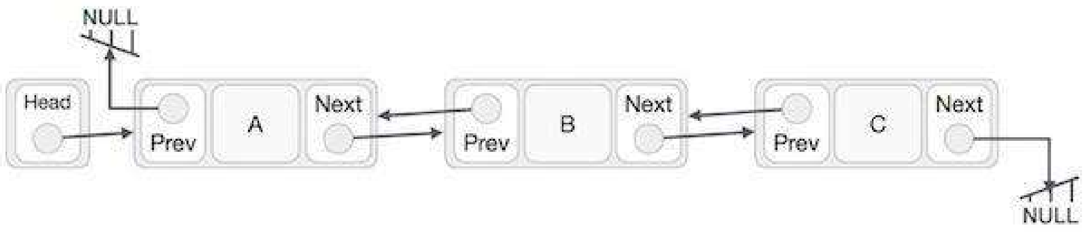
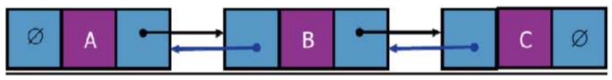


Figure 1. Doubly Linked List representation
A doubly linked list is a linked list in which every node has a next pointer and a back pointer. Every node contains the address of the next node (except the last node), and every node contains the address of the previous node (except the first node). A doubly linked list can be traversed in either direction.



Following are the basic operations supported by a list.

- **Insertion** − Adds an element at the beginning of the list.
- **Deletion** − Deletes an element at the beginning of the list.
- **Insert Last** − Adds an element at the end of the list.
- **Delete Last** − Deletes an element from the end of the list.
- **Insert After** − Adds an element after an item of the list.
- **Delete** − Deletes an element from the list using the key.

- **Display forward** − Displays the complete list in a forward manner.
- **Display backward** − Displays the complete list in a backward manner.

Following are advantages/disadvantages of doubly linked list over singly linked list
Advantages over singly linked list
A DLL can be traversed in both forward and backward direction.
The delete operation in DLL is more efficient if pointer to the node to be deleted is given.
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 a previous pointer. It is possible to implement DLL with single pointer though.
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.

Circular Linked List:
Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

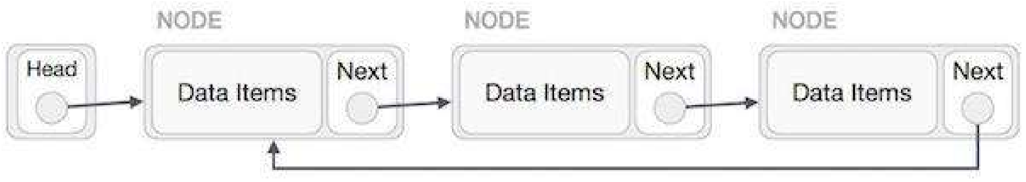


Figure 2. Singly List as Circular

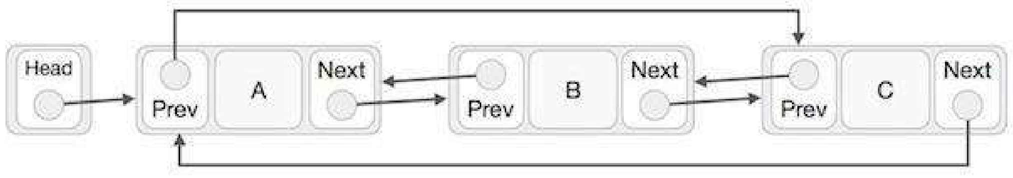


Figure 3. Doubly List as Circular

As per the above illustration, following are the important points to be considered.

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- The first link's previous points to the last of the list in case of doubly linked list.

Sample Code

Creating a Doubly Linked List

We shall take the list to be initially empty, i.e.

L–>f i r s t = L–>last = NULL;

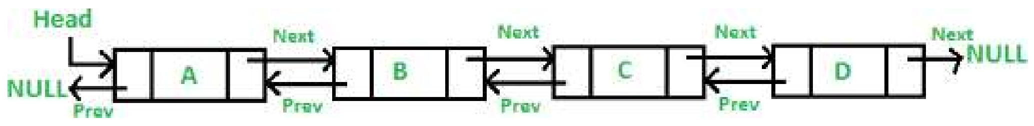
After allocating space for a new node and filling in the data (the *value* field), insertion of this node, pointed by p, is done as follows:

```
i f ( L–>f i r s t == NULL )
{ /* empty l i s t */
                L–>f i r s t = L–>last = p;
                p–>next = p–>prev= NULL;
}
else
          { /* nonempty l i s t */
              L–>last–>next = p;
              p–>prev= L–>last ;
              L–>last = p;
}
```

Lab Tasks

Doubly Linked List Tasks:

As you have learned how to insert and delete a node at different positions in the doubly linked list now you are required to perform the following tasks. Create your own class of doubly linked list class should be of template type which will have the following functions.



a) Function called **insertAtStart (int key, int val)** to add node at the beginning of list.

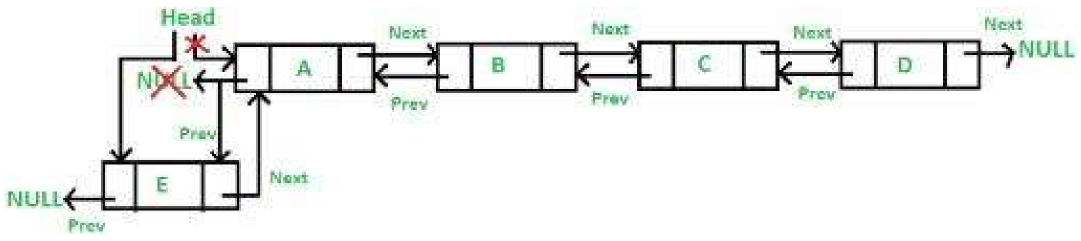
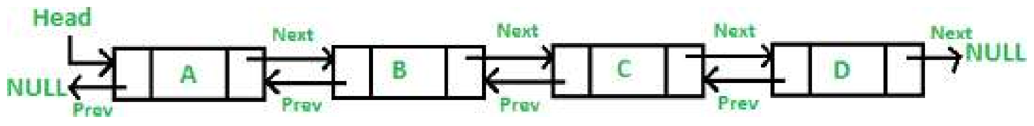


Figure 4. Insertion at beginning of the Doubly linked list

b) Function called **delete (int val)** to del node.



Let say we want to delete node C. for this we have to copy Next pointer of node C in the Next pointer of node B and Previous pointer of node C in the Previous pointer of node D. Now our list will have only A, B and D nodes.

c) Function called **ReverseDisplay ()** to display values in reverse order.

d) Function called **insertAfterSpecificNode (int key, int data)** to insert a new node after a node with a specified value in the doubly linked list.

e) Function called **totalNodesCount ()** to count the total number of nodes in a doubly linked list.

f) Function **mergeList (Node* list1, Node* list2)** to merge two sorted doubly linked lists into a single sorted doubly linked list.

g) Function called **traverseList ()** to traverse the doubly linked list in both forward and backward directions.