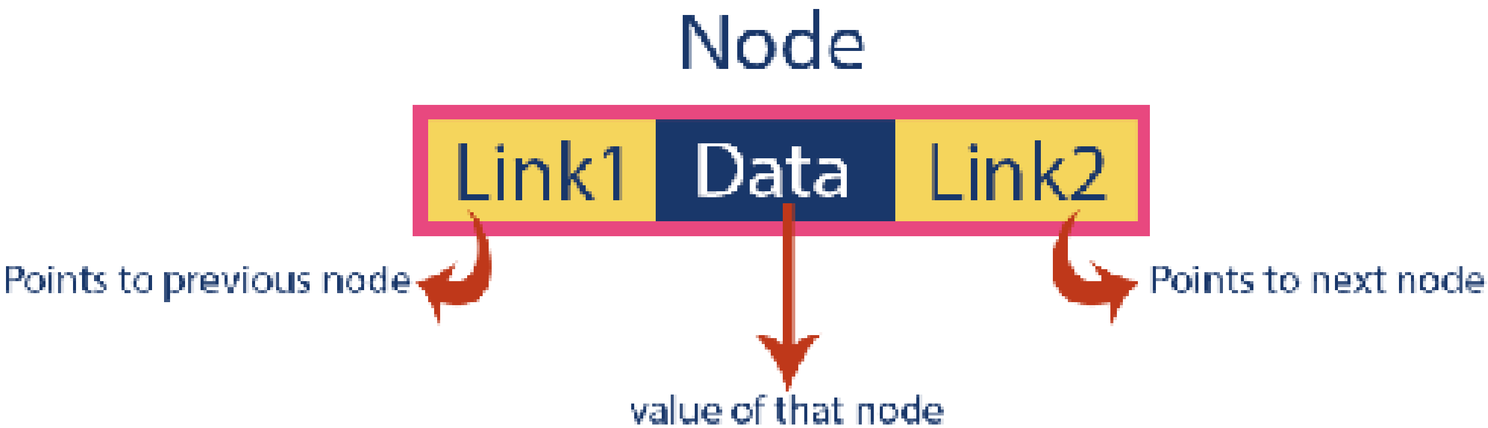
Doubly Linked Lists and Dictionary/Map/Associative Arrays

Doubly Linked Lists

What is Double Linked List?

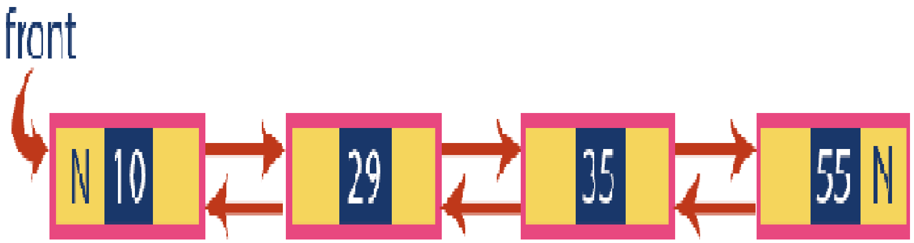
- In a single linked list, every node has a link to its next node in the sequence, so, we can traverse from one node to another node only in one direction and we can not traverse back.
- We can solve this kind of problem by using a double linked list.
- Definition: **Double linked list is a sequence of elements in which every element has links to its previous element and next element in the sequence.**
- In a double linked list, every node has a link to its previous node and next node.
- We can traverse forward by using the next field and can traverse backward by using the previous field

Representation of A double linked List



Here, **'link1'** field is used to store the address of the previous node in the sequence, **'link2'** field is used to store the address of the next node in the sequence and **'data'** field is used to store the actual value of that node.

Example



- **Importent Points to be Remembered**

> In double linked list, the first node must be always pointed by head.

> Always the previous field of the first node must be NULL.

> Always the next field of the last node must be NULL.

Operations Performed on Double Linked List

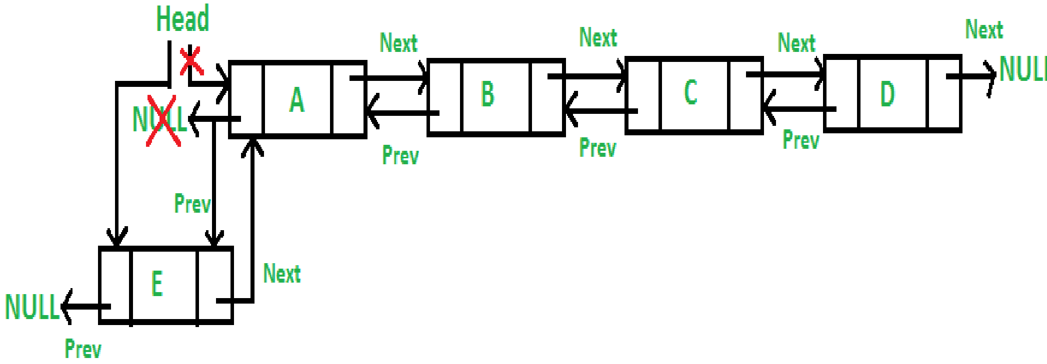
- In a double linked list, we perform the following operations...
- Insertion
- Deletion
- Display

Insertion

- A node can be added in four ways
  - At the front of the DLL
  - After a given node.
  - At the end of the DLL
  - Before a given node.

Add a node at the front

- The new node is always added before the head of the given Linked List.
- And newly added node becomes the new head of DLL.
- 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



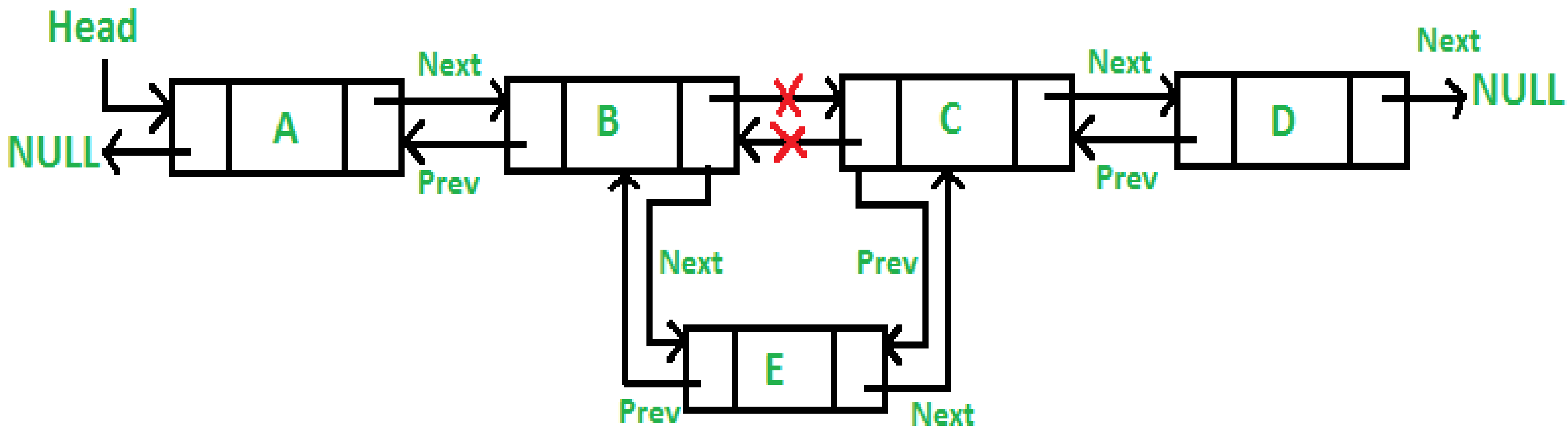
Algorithm for Inserting a node

- **Step 1 -** Create a **newNode** with given value and **newNode ŉ previous** as **NULL**.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, assign **NULL** to **newNode ŉ next** and **newNode** to **head**.
- **Step 4 -** If it is **not Empty** then, assign **head** to **newNode ŉ next** and **newNode** to **head**.

Inserting At Specific location in the list (After a Node)

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty** then, assign **NULL** to both **newNode → previous** & **newNode → next** and set **newNode** to **head**.
- **Step 4 -** If it is **not Empty** then, define two node pointers **temp1** & **temp2** and initialize **temp1** with **head**.
- **Step 5 -** Keep moving the **temp1** to its next node until it reaches to the node after which we want to insert the newNode (until **temp1 → data** is equal to **location**, here location is the node value after which we want to insert the newNode).
- **Step 6 -** Every time check whether **temp1** is reached to the last node. If it is reached to the last node then display **'Given node is not found in the list!!! Insertion not possible!!!'** and terminate the function. Otherwise move the **temp1** to next node.
- **Step 7 -** Assign **temp1 → next** to **temp2**, **newNode** to **temp1 → next**, **temp1** to **newNode → previous**, **temp2** to **newNode → next** and **newNode** to **temp2 → previous**.

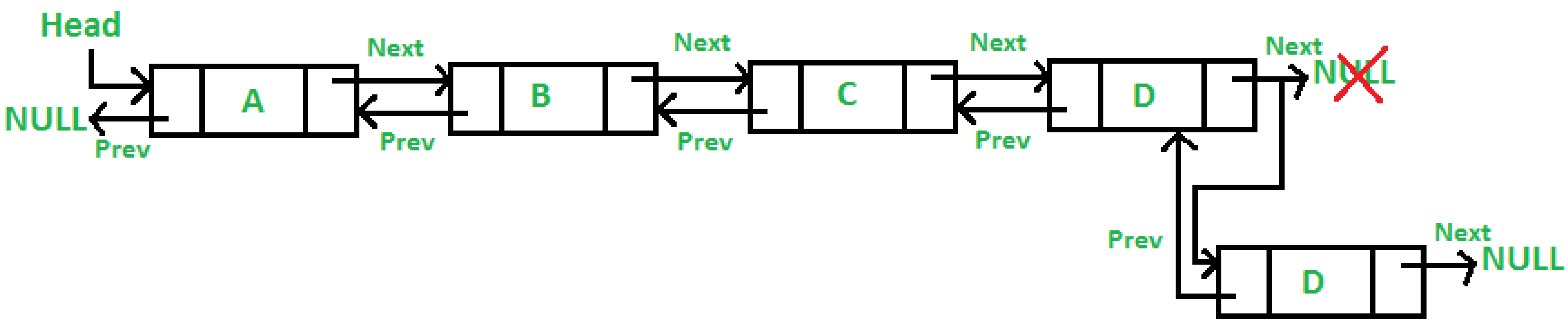
Add a Node after a Given Node.



- We make the next of node B connect to E and the previous of E connect to B.
- Then the previous of C is joined to E and the Next of E is connected to C.

Add a node at the end

- 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.



Algorithm

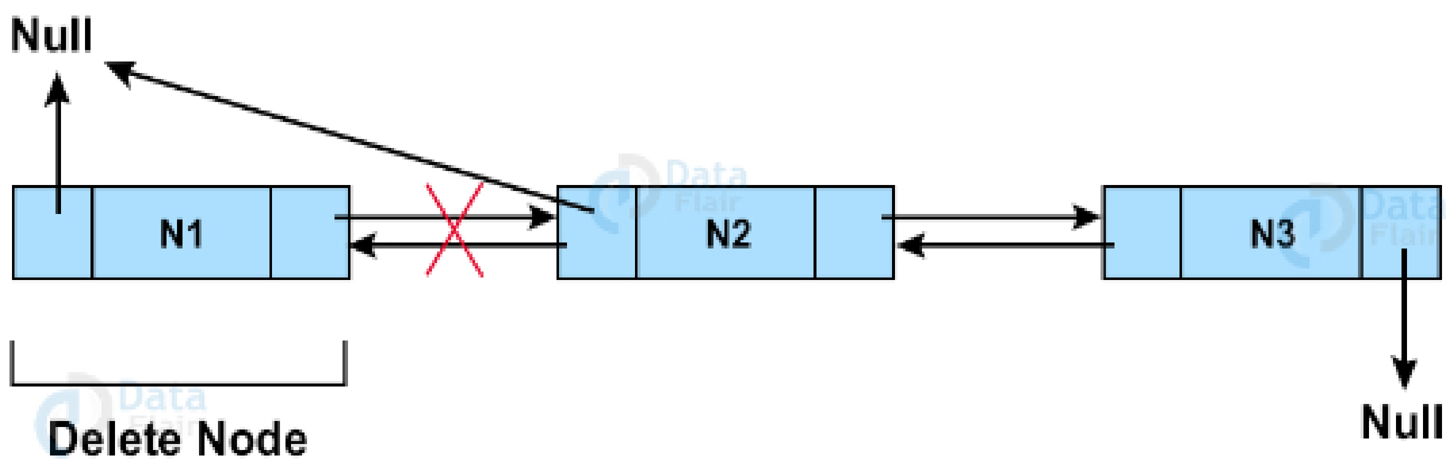
- **Step 1 -** Create a **newNode** with given value and **newNode ŉ next** as **NULL**.
- **Step 2 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 3 -** If it is **Empty**, then assign **NULL** to **newNode ŉ previous** and **newNode** to **head**.
- **Step 4 -** If it is **not Empty**, then, define a node pointer **temp** and initialize with **head**.
- **Step 5 -** Keep moving the **temp** to its next node until it reaches to the last node in the list (until **temp ŉ next** is equal to **NULL**).
- **Step 6 -** Assign **newNode** to **temp ŉ next** and **temp** to **newNode ŉ previous**.

Deletion

- In a double linked list, the deletion operation can be performed in three ways as follows...
- Deleting from Beginning of the list
- Deleting from End of the list
- Deleting a Specific Node

Deleting from Beginning of the list

- Set the Head node’s Next to point towards a null value
- Set the second Node’s Previous also to point to a null value

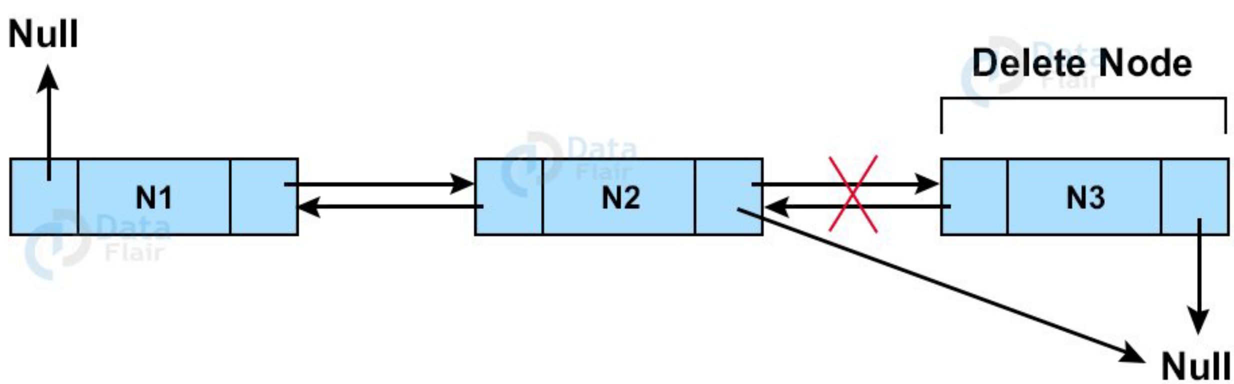


Deleting from Beginning of the list

- We can use the following steps to delete a node from beginning of the double linked list...
- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is not Empty then, define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Check whether list is having only one node (**temp ʹn previous** is equal to **temp ʹn next**)
- **Step 5 -** If it is **TRUE**, then set **head** to **NULL** and delete **temp** (Setting **Empty** list conditions)
- **Step 6 -** If it is **FALSE**, then assign **temp ʹn next** to **head**, **NULL** to **head ʹn previous** and delete **temp**.

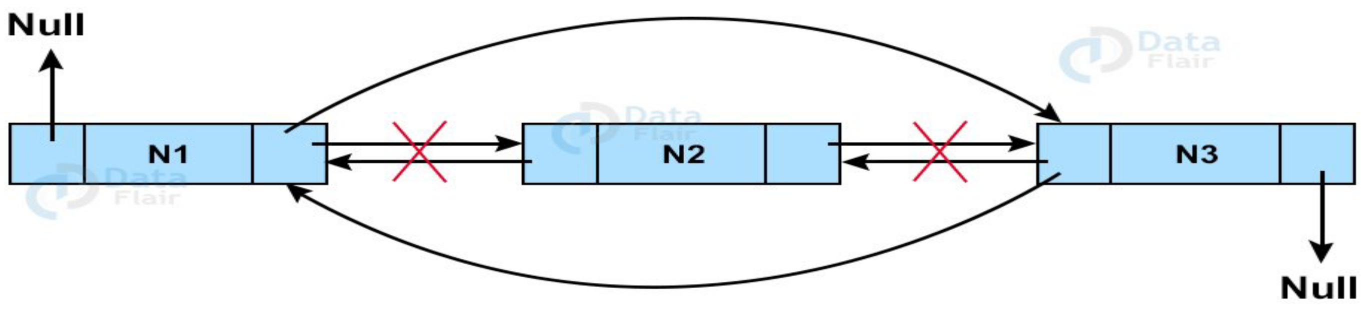
Delete node at the end of Doubly Linked List

- Point next pointer of the second last node and previous pointer of the last node to null.



Delete node at any position in Doubly Linked List

- If you want to delete the node at the nth position, point the next of (n-1)th to (n+1)th node and point the previous of (n+1)th node to the(n-1)th node. Point previous and next of nth node to null.



- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty** then, display **'List is Empty!!! Deletion is not possible'** and terminate the function.
- **Step 3 -** If it is not Empty, then define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Keep moving the **temp** until it reaches to the exact node to be deleted or to the last node.
- **Step 5 -** If it is reached to the last node, then display **'Given node not found in the list! Deletion not possible!!!'** and terminate the fuction.
- **Step 6 -** If it is reached to the exact node which we want to delete, then check whether list is having only one node or not
- **Step 7 -** If list has only one node and that is the node which is to be deleted then set **head** to **NULL** and delete **temp** (**free(temp)**).

Algorithm for deleting a node from Specific Location

- **Step 8 -** If list contains multiple nodes, then check whether **temp** is the first node in the list (**temp == head**).
- **Step 9 -** If **temp** is the first node, then move the **head** to the next node (**head = head ŉ next**), set **head** of **previous** to **NULL** (**head ŉ previous = NULL**) and delete **temp**.
- **Step 10 -** If **temp** is not the first node, then check whether it is the last node in the list (**temp ŉ next == NULL**).
- **Step 11 -** If **temp** is the last node then set **temp** of **previous** of **next** to **NULL** (**temp ŉ previous ŉ next = NULL**) and delete **temp** (**free(temp)**).
- **Step 12 -** If **temp** is not the first node and not the last node, then set **temp** of **previous** of **next** to **temp** of **next** (**temp ŉ previous ŉ next = temp ŉ next**), **temp** of **next** of **previous** to **temp** of **previous** (**temp ŉ next ŉ previous = temp ŉ previous**) and delete **temp** (**free(temp)**).

Displaying a Double Linked List

- We can use the following steps to display the elements of a double linked list...
- **Step 1 -** Check whether list is **Empty** (**head** == **NULL**)
- **Step 2 -** If it is **Empty**, then display **'List is Empty!!!'** and terminate the function.
- **Step 3 -** If it is not Empty, then define a Node pointer **'temp'** and initialize with **head**.
- **Step 4 -** Display **'NULL <--- '**.
- **Step 5 -** Keep displaying **temp ʼn data** with an arrow (**<===>**) until **temp** reaches to the last node
- **Step 6 -** Finally, display **temp ʼn data** with arrow pointing to **NULL** (**temp ʼn data ---> NULL**).