Doubly Linked List

A doubly linked list is a linear data structure, in which the elements are stored in the form of a node. Each node contains three sub-elements. A data part that stores the value of the element, the previous part that stores the pointer to the previous node, and the next part that stores the pointer to the next node

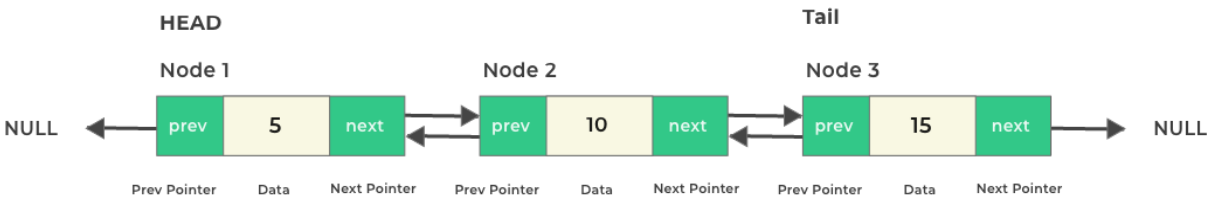
Structure of a Doubly Linked List in Java

The difference between a Linked List and a Doubly Linked is that singly linked list just has reference to the next node in the chain while doubly linked list has reference to both previous and next node in the chain

Components of doubly linked list are –

- Data
- Reference to the Next node
- Reference to the Previous node

Doubly Linked List allows traversal in both forward and backward direction as for any node both previous and next nodes are known.



Operations on a Doubly Linked List

Following operations can be performed on a doubly linked list:-

- **Insertion**
  - Insertion at the beginning.
  - Insertion at the end.
  - Insertion at a specific position.
- **Deletion**
  - Deletion from the beginning.

- Deletion from the end.
- Deletion from a specific position.

```
package linklist;

class DoublyLinkedList {
  Node head;

  // Node Class
  class Node {
    int data;
    Node next, prev;

    Node(int x) // parameterized constructor
    {
      data = x;
      next = null;
      prev = null;
    }
  }

  // inserts at first position
  public void insertNode(int data) {
    // Creating newNode memory & assigning data value
    Node newNode = new Node(data);

    newNode.next = head;
    newNode.prev = null;

    // if DLL had already >=1 nodes
    if (head != null)
      head.prev = newNode;

    // changing head to this
    head = newNode;
    System.out.println(newNode.data + " inserted");
  }

  public void display() {
    Node node = head;
    Node end = null;
    // as linked list will end when Node reaches Null

    System.out.print("\nIn forward: ");
    while (node != null) {
      System.out.print(node.data + " ");
      end = node;
      node = node.next;
    }
    System.out.print("\nIn backward: ");

    while (end != null) {
      System.out.print(end.data + " ");
      end = end.prev;
    }
```

```
    System.out.println("\n");
  }

  public static void main(String args[]) {
    DoublyLinkedList dll = new DoublyLinkedList();

    dll.insertNode(26);
    dll.insertNode(25);
    dll.insertNode(24);

    dll.display();

    dll.insertNode(23);
    dll.insertNode(22);
    dll.insertNode(21);
    dll.insertNode(20);

    dll.display();
  }
}
```

Insertion in Doubly Linked List

In a doubly Linked List insertion can be done at the following positions –

- At the beginning
- At the end
- After a given node

Insertion at the Beginning of a Doubly Linked List

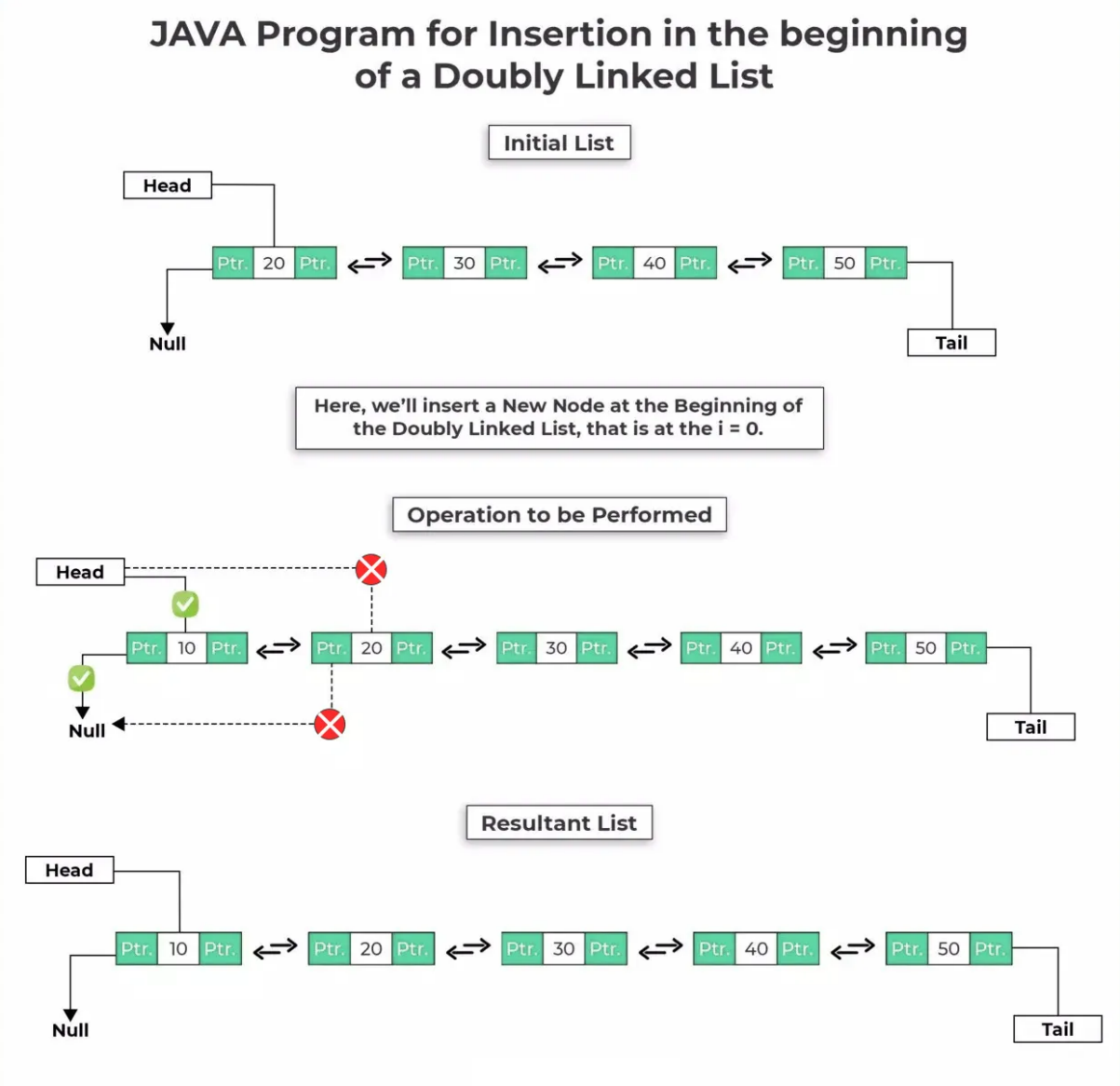
Adding or inserting a node in the beginning in doubly linked list is almost similar as the process of adding a node in singly linked list . The only difference is that we have an extra pointer (previous node) to be redirected.

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to insert a node in the beginning of the Doubly Linked List, we'll have to store and redirect various links of the Linked List.
- First of all the Head will now store the address of the space where the data of the New Node is stored.
- Now Since the New Node is going to be the first Node of the Linked List. So, the Previous Pointer of the New Node will point to Null.

- Then the Next Pointer of the New Node will now point to the Previously First Node of the List.
- Now, at last the Previous Pointer of the Previously First node of the list will be directed towards the New Node that is being Inserted.

Algorithm

- AppendStart(int data)
- Node newNode = new Node(data)
- IF HEAD == NULL
  - newNode HEAD = TAIL = newNode
  - HEAD.previous = NULL
  - TAIL.next = NULL
- ELSE
  - HEAD.previous = newNode
  - newNode.next = HEAD
  - newNode.previous = NULL
  - HEAD = newNode



Insertion in the End of a Doubly Linked List

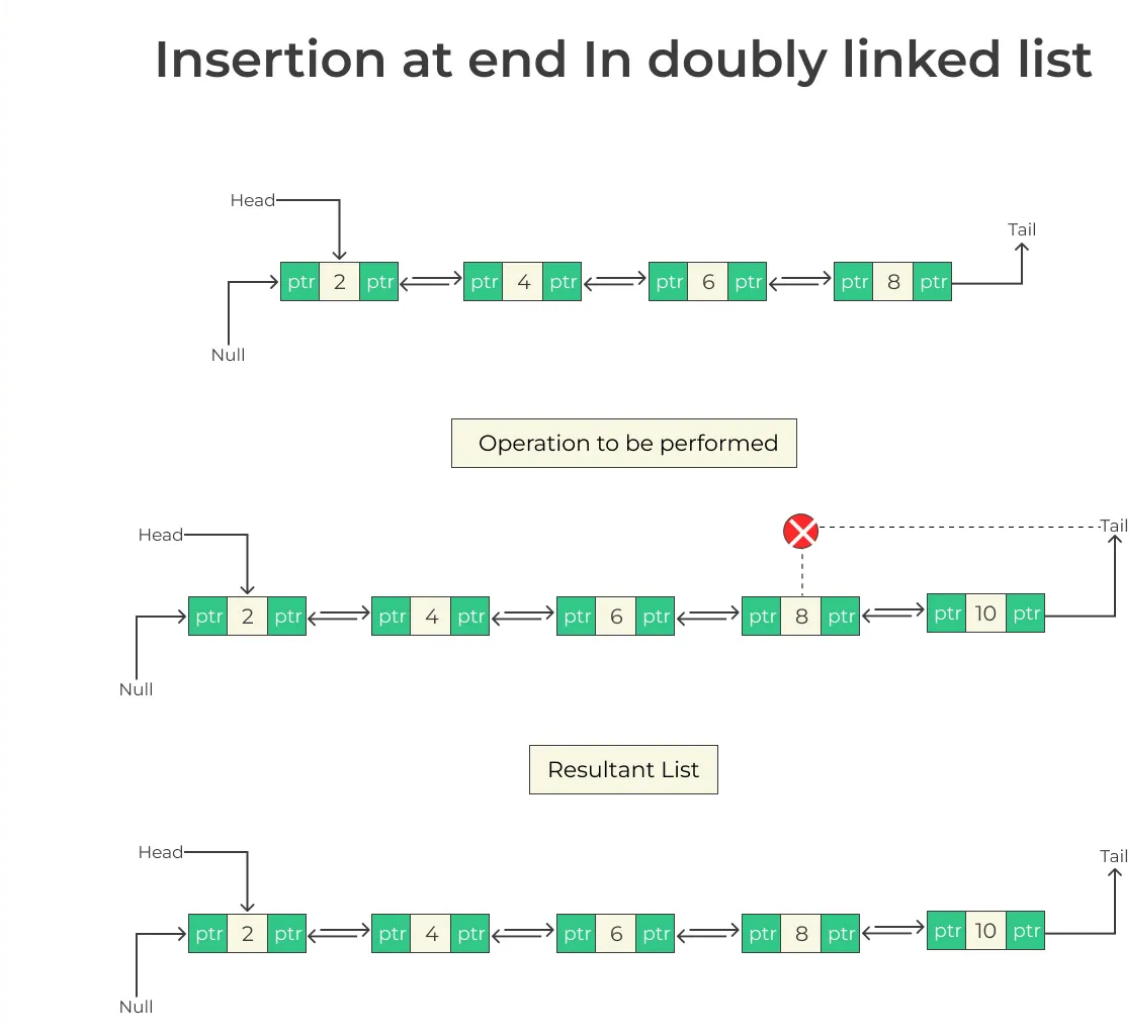
To insert a new node at the end of a linked list we have to check that the list is empty or not. If the list is empty both head and tail points towards the newly added node. If it is not empty, in such case add a new node at the end and make sure that tail points towards the newly added node.

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to insert a node in the end of the Doubly Linked List, we'll have to store and redirect various links of the Linked List.
- First of all the next pointer of the previously last Node will store the address of the New Node that is being Inserted in the Linked List.

- Now Since the New Node is going to be the last Node of the Linked List. So, the Next Pointer of the New Node will point to Tail.
- Then the Previous Pointer of the New Node will now point to the Previously Last Node of the Linked List.

Algorithm

- appendAtEnd (data)
- if(HEAD == NULL)
    - HEAD = TAIL = newNode
    - HEAD.prev = NULL
    - TAIL.next = NULL
- Else
    - TAIL.next = newNode
    - newNode.prev = TAIL
    - TAIL = newNode
    - TAIL.next = NULL



Insertion at the n^{th} Position of a Doubly Linked List

The Process of insertion in a doubly linked list is somewhat similar to the process of insertion in a Singly Linked List, the difference here is just that here we have a extra pointer (Previous) that needs to be directed to the address of the node lying before the node that is being Inserted.

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to insert a node at the Nth Position of the Doubly Linked List, we'll have to store and redirect various links of the Linked List.
- First of all the address stored in the next pointer of the (n-1)th node of the List will now store the address of the New Node that is being inserted.
- Now the address stored in the Previous Pointer of the (n+1)th node of the Linked List will also be re-directed to the address

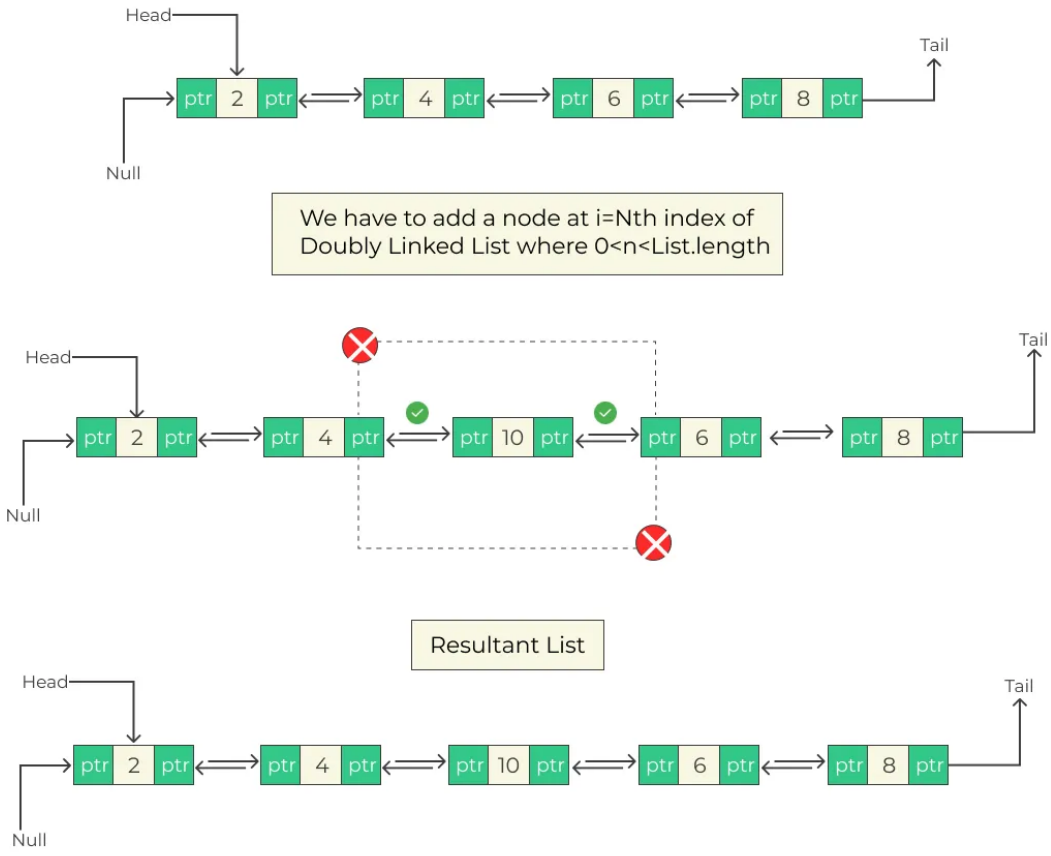
of the New Node being inserted.

- Now, at last the Previous
  Pointer of the New Node will be directed towards the address of the node at (n-1)th position and the Next Pointer of the New Node will be
  directed towards the address of the node at (n+1)th position.

Algorithm

- Node node=new Node()
- node.data=data
- node.nextNode=null
- if(this.head==null)
- if(location!=0)
- return
- else
- this.head=node
- if(head!=null&&location==0)
- node.nextNode=this.head
- this.head=node
- return
- Node curr=this.head
- Node prev =Nnull
- while(i=0)
- tempNode=tempNode.nextNode

Insertion at Nth position in a doubly linked list




```
package linklist;

public class DoublyLinkedList_Insert {
  Node head;
  // not using parameterized constructor would by default
  // force head instance to become null
  // Node head = null; // can also do this, but not required

  // Node Class
  class Node {
    int data;
    Node next, prev;

    Node(int x) // parameterized constructor
    {
      data = x;
      next = null;
      prev = null;
    }
  }

  public void insertBeginning(int data) {
```

```
    // Creating newNode memory & assigning data value
    Node freshNode = new Node(data);

    freshNode.next = head;
    freshNode.prev = null;

    // if DLL had already >=1 nodes
    if (head != null)
      head.prev = freshNode;

    // changing head to this
    head = freshNode;
    System.out.println(data+" is inserted at the Beginning");
  }

  public void insertEnd(int data) {
    // Creating newNode memory & assigning data value
    Node freshNode = new Node(data);

    // assign data
    // since this will be the last node its next will be NULL
    freshNode.next = null;

    // if we are entering the first node
    if (head == null) {
      head = freshNode;
      freshNode.prev = null;
      System.out.println(data+" is inserted at the First Node");
      return;
    }

    Node last = head;

    // traverse to the current last node
    while (last.next != null)
      last = last.next;

    // assign current last node's next to this new node
    // assign new node's previous to this last node
    last.next = freshNode;
    freshNode.prev = last;
    // new_node becomes the last node now]
    System.out.println(data+" is inserted at the End Node");

  }

  public void insertAfterPosition(int n, int data) {
    int len = getLength(head);
    int pos=n;

    // if insertion position is 0 means entering at start
    if (n == 0) {
      System.out.println("It is the First Position");
      insertBeginning(data);
      return;
    }
    // means inserting after last item
    if (n == len) {
```

```
      System.out.println("It is the Last Position");
      insertEnd(data);
      return;
    }

    // else insertion will happen somewhere in the middle

    if (n < 1 || len < n)
      System.out.println("Invalid position");
    else {
      Node freshNode = new Node(data);
      // can remove null assignments also (constructor takes care of these)
      // added just for explanation purpose

      freshNode.next = null;

      freshNode.prev = null;
      // nthNode used to traverse the Linked List

      Node nthNode = head;

      // traverse till the nth node

      while (--n > 0) {
        nthNode = nthNode.next;
      }

      Node nextNode = nthNode.next; // (n+1)th node

      // assigning (n+1)th node's previous to this new node
      nextNode.prev = freshNode;

      // new_node's next assigned to (n+1)th node
      freshNode.next = nextNode;
      // new_node's previous assigned to nth node
      freshNode.prev = nthNode;

      // assign nth node's next to new_node
      nthNode.next = freshNode;

      System.out.println(data+" is Inserted at the "+(pos+1)+" Postion");
    }
  }

  public void printList() {
    Node node = head;
    Node end = null;
    // as linked list will end when Node reaches Null

    System.out.print("\nIn forward: ");
    while (node != null) {
      System.out.print(node.data + " ");
      end = node;
      node = node.next;
    }
    System.out.print("\nIn backward: ");

    while (end != null) {
```

```
      System.out.print(end.data + " ");
      end = end.prev;
    }
    System.out.println();
  }

  public int getLength(Node node) {
    int size = 0;
    // traverse to the last node each time incrementing the size
    while (node != null) {
      node = node.next;
      size++;
    }
    return size;
  }

  public static void main(String args[]) {
    DoublyLinkedList_Insert doublylist = new DoublyLinkedList_Insert();

    doublylist.insertBeginning(3);
    doublylist.insertBeginning(2);
    doublylist.insertBeginning(1);
    doublylist.insertBeginning(4);
    doublylist.insertBeginning(5);

    doublylist.printList();

    doublylist.insertEnd(30);
    doublylist.insertEnd(20);
    doublylist.insertEnd(10);
    doublylist.insertEnd(40);
    doublylist.insertEnd(50);

    doublylist.printList();

    // Inserts after 4th position
    doublylist.insertAfterPosition(4,123);

    doublylist.printList();

    doublylist.insertAfterPosition(5,234);

    doublylist.printList();

  }
}
```

Deletion in Doubly Linked List

We have three kinds of deletion in singly linked list in the same way we have three kinds of deletion in doubly linked list are as follows:

- Deletion at the beginning .

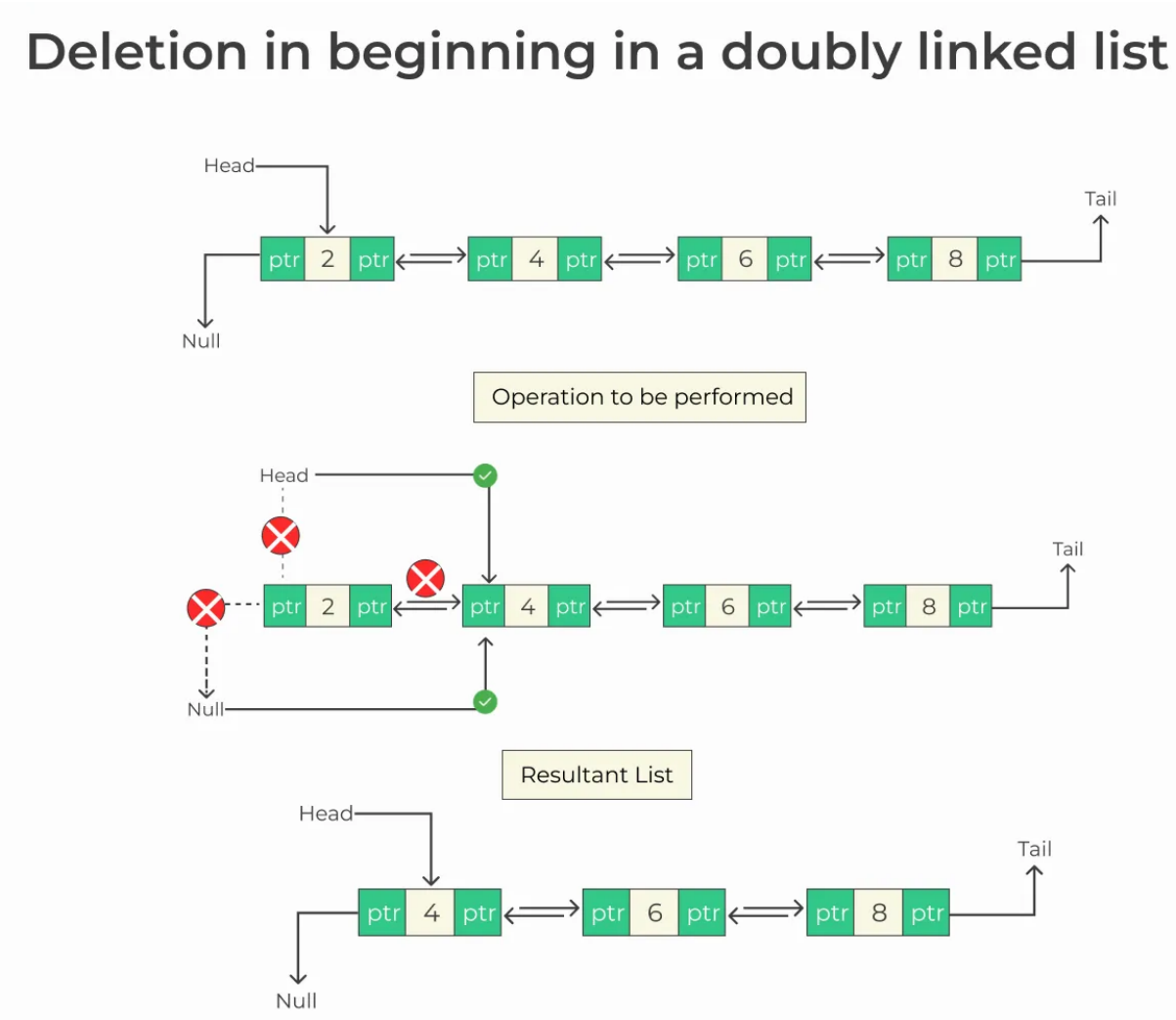
- Deletion at n^{th} node/in middle .
- Deletion at the end .

Deletion from the Beginning of a Doubly Linked List

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to Delete a node from the end of the Doubly Linked List, we'll have to create and redirect multiple links in the Linked List.
- Foremost, the link between the First Node and the head and the link between the Previous Pointer of the First Node and the null will also be broken.
- Now Since the Second Node is going to be the First Node of the Linked List. So, the Head will now Point to the address of the Second Node.
- Then the Previous Pointer of the Second Node will now point to Null.

Algorithm

- IF(HEAD == NULL)
- RETURN
- ELSE IF(HEAD != TAIL)
    - TAIL = TAIL.PREV
    - TAIL.NEXT = NULL
- ELSE
    - HEAD = TAIL = NULL



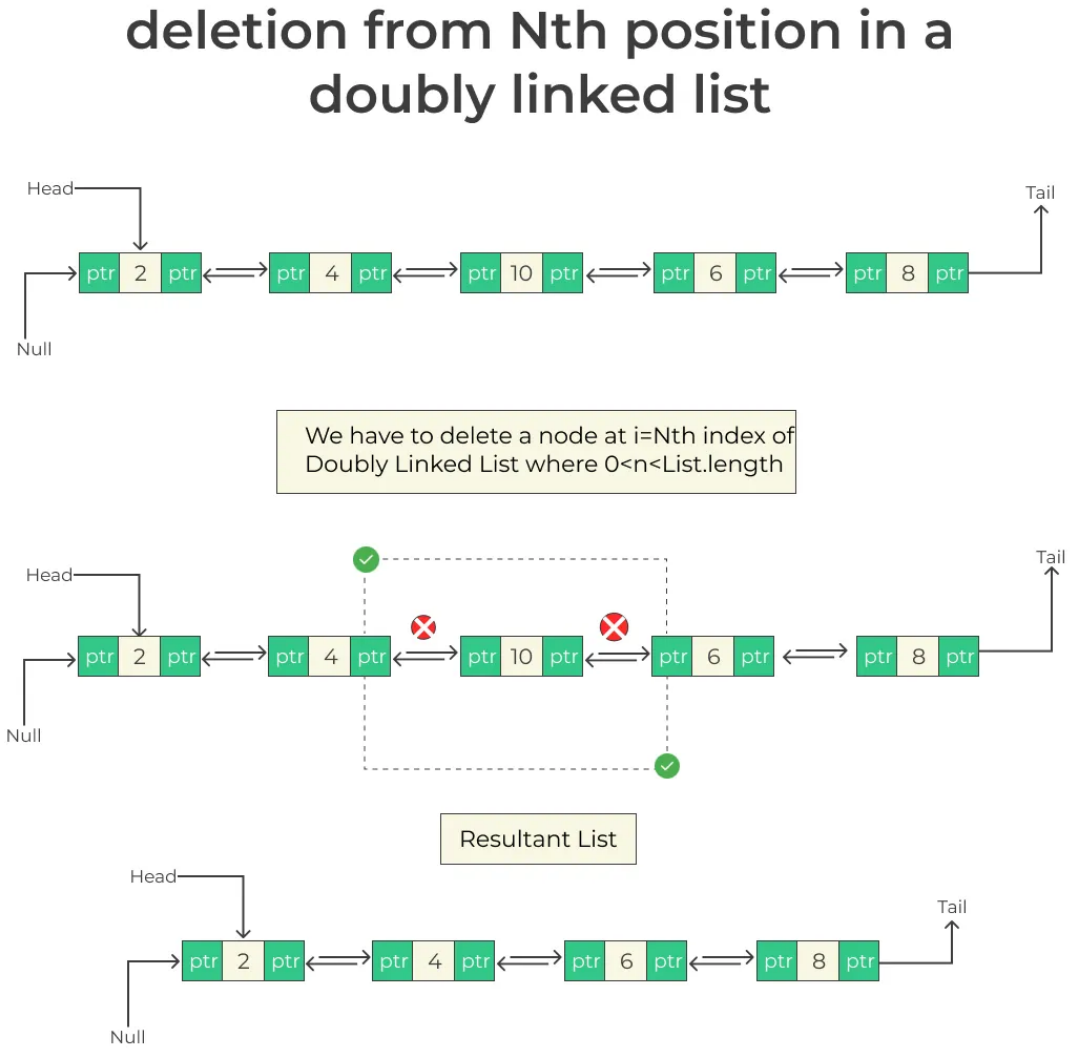
Deletion from n^{th} Position of a Doubly Linked List

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to delete a node from Nth position of the Doubly Linked List, we'll have to delete and redirect various links of the Linked List.
- First of all the next pointer of the (n-1)th Node of the Linked List will now point to the address of the (n+1)th node of the Doubly Linked List.
- Now the Previous Pointer of the (n+1)th Node of the Linked List will now be re-directed to the address of (n-1)th node of the List.

Algorithm

- IF(HEAD == NULL)
  - RETURN

- ELSE
  - NODE CURRENT = HEAD;
  - INT POS =N;
- FOR(INT I = 1; I < POS; I++)
  - CURRENT = CURRENT.NEXT
- IF(CURRENT == HEAD)
  - HEAD = CURRENT.NEXT
- ELSE IF(CURRENT == TAIL)
  - TAIL = TAIL.PREV
- ELSE
  - CURRENT.PREV.NEXT = CURRENT.NEXT
  - CURRENT.NEXT.PREV = CURRENT.PREV
- CURRENT = NULL



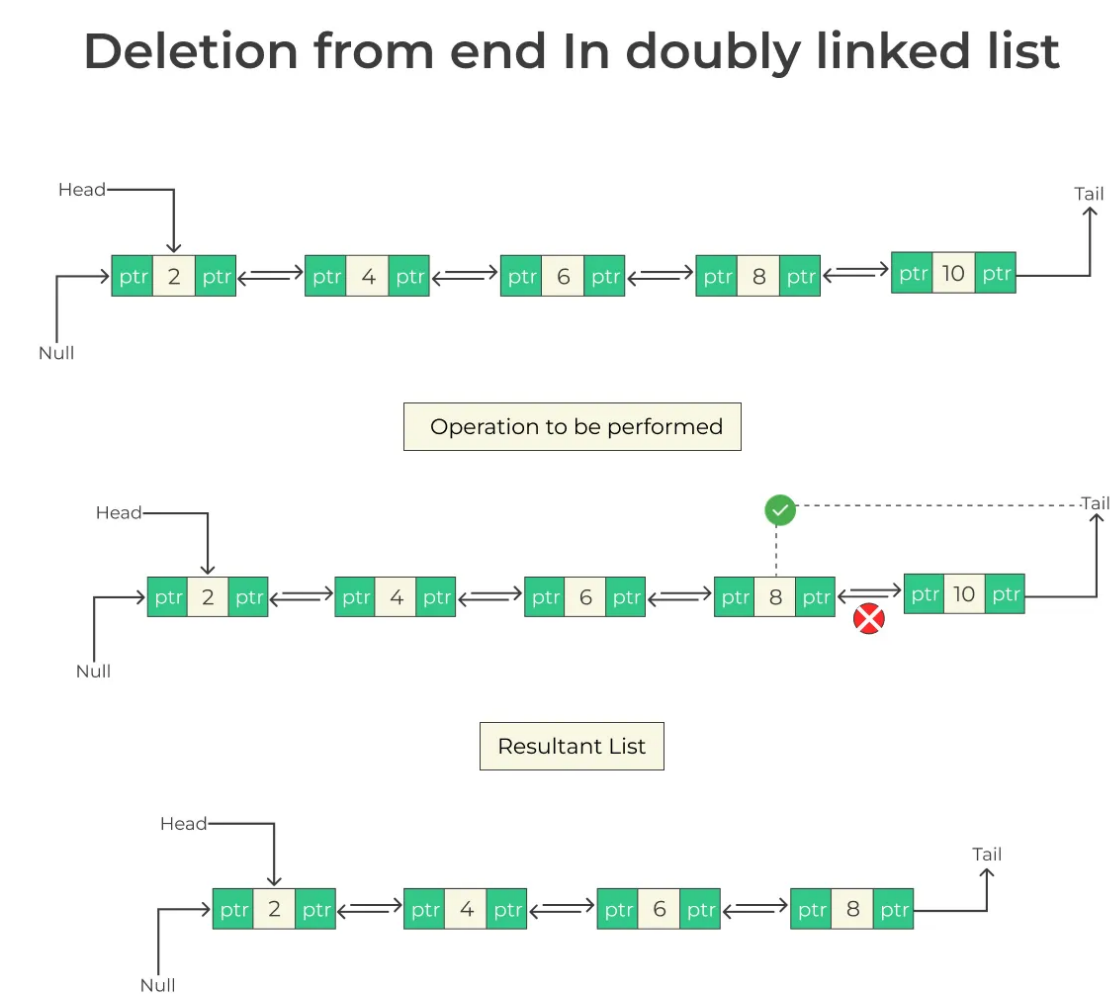
Deletion from End of a Doubly Linked List

- Check for the presence of Node in the List, if there exists some Nodes, Continue.
- Now, to Delete a node from the end of the Doubly Linked List, we'll have to redirect links of the Linked List.
- Let the length of the List = i.
- First of all the next pointer of the (i-1)th node of the Linked List will now be re-directed towards the Tail of the Linked List.

Algorithm

- IF(HEAD == NULL)
  - RETURN

- ELSE IF(HEAD != TAIL)
  - HEAD = HEAD.NEXT
  - TAIL.NEXT = NULL
- ELSE
  - HEAD = TAIL = NULL




```
package linklist;

public class DoublyLinkedList_Delete {
  Node head;
  // not using parameterized constructor would by default
  // force head instance to become null
  // Node head = null; // can also do this, but not required
```

```
// Node Class
class Node {
  int data;
  Node next, prev;

  Node(int x) // parameterized constructor
  {
    data = x;
    next = null;
    prev = null;
  }
}

public void insertBeginning(int data) {
  // Creating newNode memory & assigning data value
  Node freshNode = new Node(data);

  freshNode.next = head;
  freshNode.prev = null;

  // if DLL had already >=1 nodes
  if (head != null)
    head.prev = freshNode;

  // changing head to this
  head = freshNode;
}

// function for deleting first node
public void delete_Start() {
  if (head == null) {
    System.out.println("List is empty");
    return;
  } else {
    // testing for the presence of a single node in the list, If not, Then head and
    // tail will be re-directed
    if (head != null) {
      head = head.next;
    }
    // if only one node exist both head and tail will be redirected to null
    else {
      head = null;
    }
  }
}

// function for deleting last node
public void delete_Last() {
  if (head == null) {
    return;
  } else {
    if (head != null) {
      Node temp = head;

      while (temp.next != null)
        temp = temp.next;
      temp = temp.prev;
      temp.next = null;
```

```
    } else {
      head = null;
    }
  }
}

// function for deleting Nth node
public void deleteNth(int n) {
  if (head == null) {
    return;
  } else {
    Node current = head;
    int pos = n;
    for (int i = 1; i < pos; i++) {
      current = current.next;
    }
    if (current == head) {
      head = current.next;
    } else if (current == null) {
      current = current.prev;
    } else {
      current.prev.next = current.next;
      current.next.prev = current.prev;
    }
    // Delete the middle node
    current = null;
  }
}

void printList() {
  // Node current will point to head
  Node curr = head;
  if (head == null) {
    System.out.println("List is empty");
    return;
  }
  while (curr != null) {
    // Prints each node by increasing order of the pointer
    System.out.print(curr.data + " ");
    curr = curr.next;
  }
  System.out.println();
}

public static void main(String args[]) {
  DoublyLinkedList_Delete doublylist = new DoublyLinkedList_Delete();

  doublylist.insertBeginning(3);
  doublylist.insertBeginning(2);
  doublylist.insertBeginning(1);
  doublylist.insertBeginning(4);
  doublylist.insertBeginning(7);

  System.out.println("List before deletion : ");
  doublylist.printList();

  doublylist.delete_Start();
```

```
    System.out.println("List after deleting first node : ");
    doublylist.printList();

    doublylist.delete_Last();
    System.out.println("List after deleting last node : ");
    doublylist.printList();

    doublylist.deleteNth(2);
    System.out.println("List after deleting 2nd node : ");
    doublylist.printList();

  }

}
```