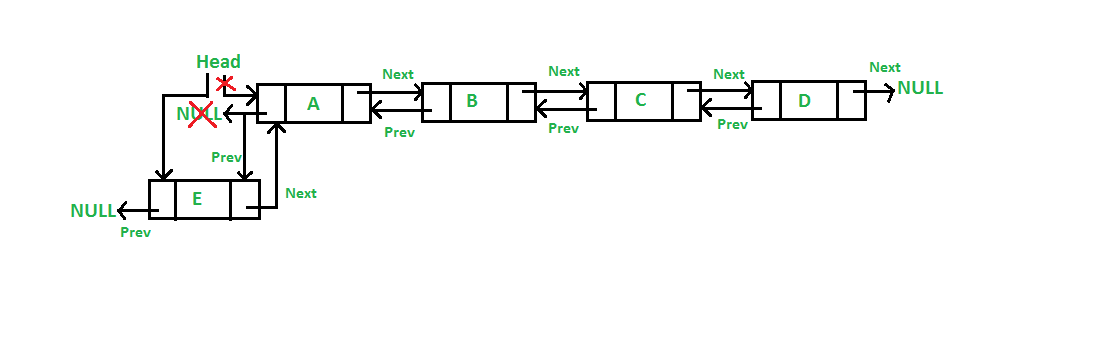
1. **Insertion in DLL**



1. **Add a node at the front**

****

/\* 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 \*/

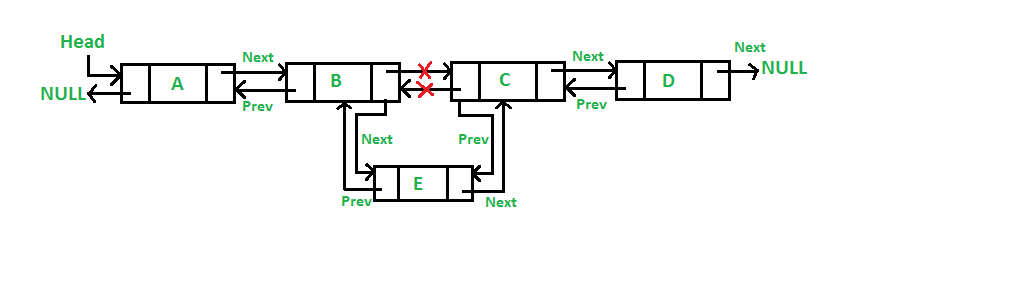
    if ((\*head\_ref) != NULL)

        (\*head\_ref)->prev = new\_node;

    /\* 5. move the head to point to the new node \*/

    (\*head\_ref) = new\_node;

### ****b) Add a node after a given node****

****

/\*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;

    /\* 7. Change previous of new\_node's next node \*/

    if (new\_node->next != NULL)

        new\_node->next->prev = new\_node;

### ****c) Add a node at the end****

### dll_add_end

/\* 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;

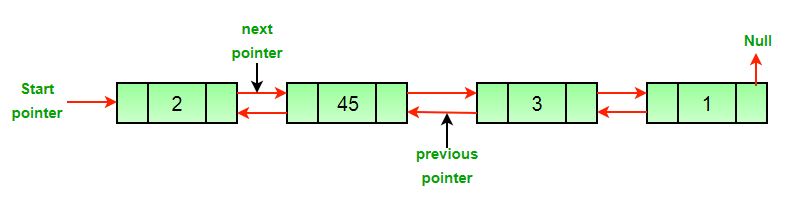
    /\* 6. Change the next of last node \*/

    last->next = new\_node;

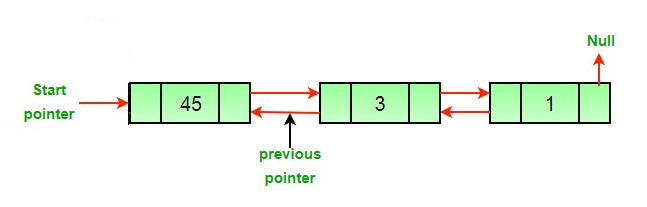
    /\* 7. Make last node as previous of new node \*/

    new\_node->prev = last;

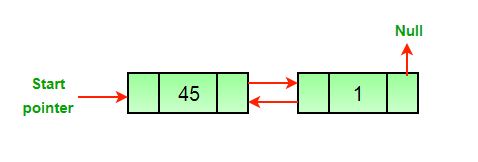
1. **Deletion in DLL**



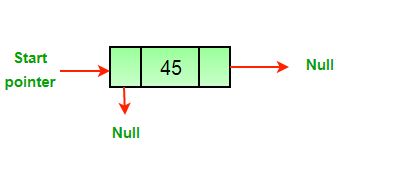
* **After the deletion of the head node.**



* **After the deletion of the middle node.**



* **After the deletion of the last node.**



**Algorithm:**

* Let the node to be deleted be *del*.
* If node to be deleted is head node, then change the head pointer to next current head.

if *headnode* == *del* then

*headnode* = *del*.nextNode

* Set prev of next to del, if next to del exists.

if *del*.nextNode != *none*

*del*.nextNode.previousNode = *del*.previousNode

* Set next of previous to del, if previous to del exists.

if *del*.previousNode != *none*

*del*.previousNode.nextNode = *del*.next

1. **Find Size of DLL and CLL**

DLL

* *Initialize size to 0.*
* *Initialize a node pointer, temp = head.*
* *Do following while temp is not NULL*
  + *temp = temp -> next*
  + *size++;*
* *Return size.*

CLL

int countNodes(Node\* head)

{

    Node\* temp = head;

    int result = 0;

    if (head != NULL) {

        do {

            temp = temp->next;

            result++;

        } while (temp != head);

    }

    return result;

}

1. **To check if a linked list is Circular Linked List or not.**

int isCircular(struct Node\* head)

{

    // If linked list is empty it is circular

    if (head == NULL)

        return 1;

    struct Node\* ptr;

    ptr = head->next;

    // Traversing linked list till last node

    while (ptr != NULL && ptr != head)

        ptr = ptr->next;

    // Condition for circular linked list

    return (ptr == head);

}

1. **To check if a doubly linked list of characters is palindrome or not.**

bool isPalindrome(struct Node \*left)

{

    if (left == NULL)

       return true;

    // Find rightmost node

    struct Node \*right = left;

    while (right->next != NULL)

        right = right->next;

    while (left != right)

    {

        if (left->data != right->data)

            return false;

        left = left->next;

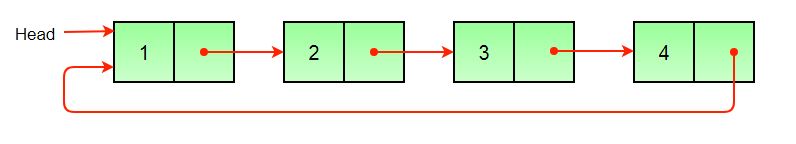
        right = right->prev;

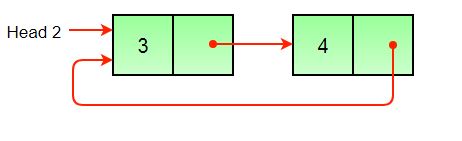
    }

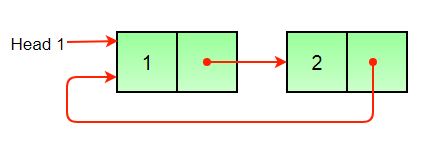
    return true;

}

1. **Split-a-circular-linked-list-into-two-halves**





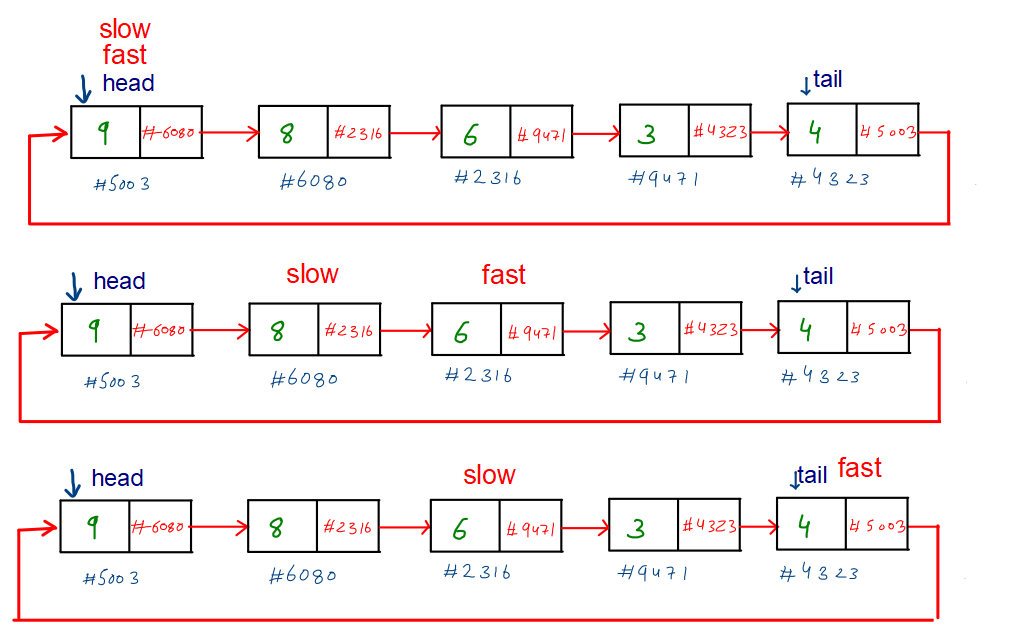


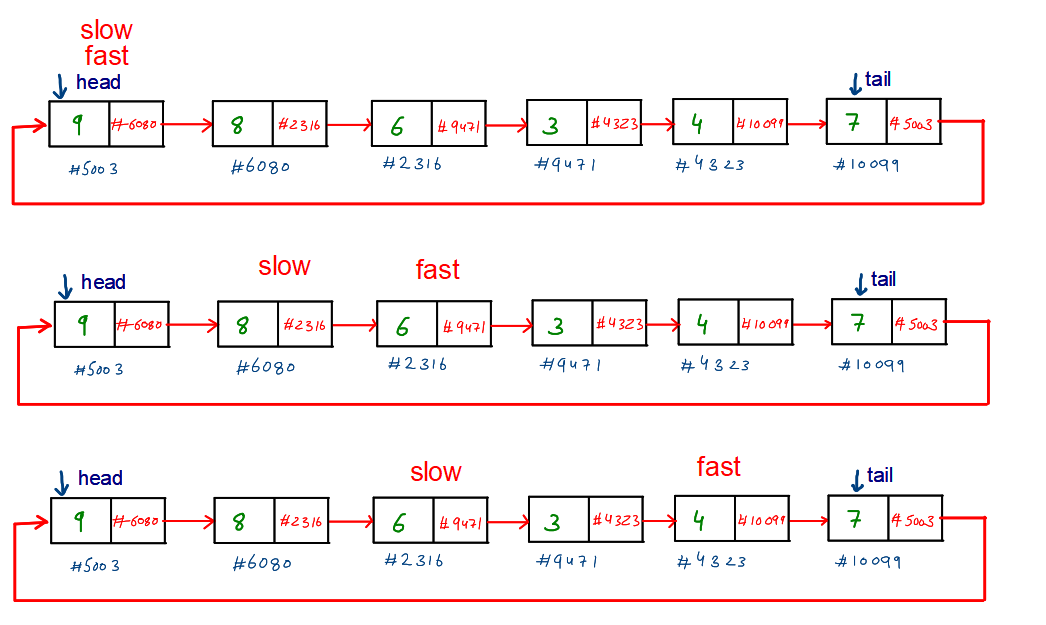
1) Store the mid and last pointers of the circular linked list using tortoise and hare algorithm.

2) Make the second half circular.

3) Make the first half circular.

4) Set head (or start) pointers of the two linked lists





void splitList(Node \*head, Node \*\*head1\_ref,

                           Node \*\*head2\_ref)

{

    Node \*slow\_ptr = head;

    Node \*fast\_ptr = head;

    if(head == NULL)

        return;

    /\* If there are odd nodes in the circular list then

       fast\_ptr->next becomes head and for even nodes

       fast\_ptr->next->next becomes head \*/

    while(fast\_ptr->next != head &&

          fast\_ptr->next->next != head)

    {

        fast\_ptr = fast\_ptr->next->next;

        slow\_ptr = slow\_ptr->next;

    }

    /\* If there are even elements in list

       then move fast\_ptr \*/

    if(fast\_ptr->next->next == head)

        fast\_ptr = fast\_ptr->next;

    /\* Set the head pointer of first half \*/

    \*head1\_ref = head;

    /\* Set the head pointer of second half \*/

    if(head->next != head)

        \*head2\_ref = slow\_ptr->next;

    /\* Make second half circular \*/

    fast\_ptr->next = slow\_ptr->next;

    /\* Make first half circular \*/

    slow\_ptr->next = head;

}