Delete Kth of a Circular Linked List

Given a Circular Linked List. The task is to write program to delete node from kth index from the list.

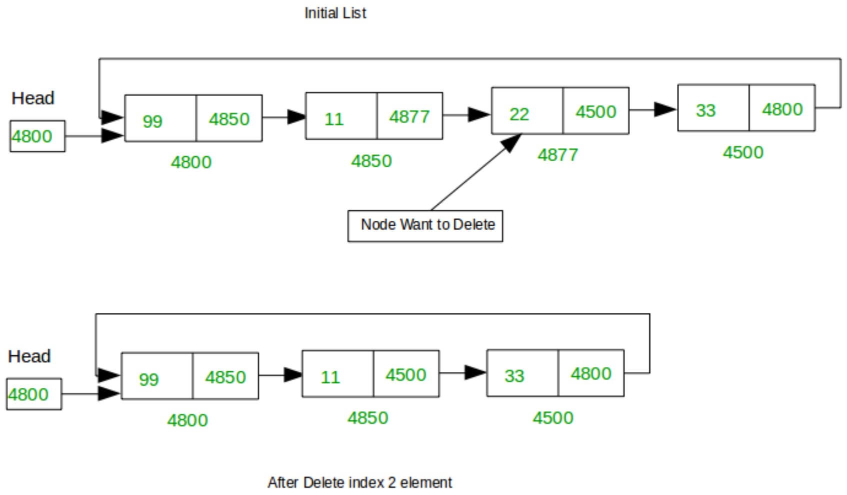
Deleting nodes at given index in the Circular linked list

Examples:

```
Input: 99->11->22->33->44->55->66
        Index= 4
Output: 99->11->22->33->55->66

Input: 99->11->22->33->44->55->66
        Index= 2
Output: 99->11->33->44->55->66
```

Note: 0-based indexing is considered for the list.



Approach:

1. First, find the length of the list. That is, the number of nodes in the list.
2. Take two pointers previous and current to traverse the list. Such that previous is one position behind the current node.

3. Take a variable count initialized to 0 to keep track of the number of nodes traversed.
4. Traverse the list until the given index is reached.
5. Once the given index is reached, do **previous->next = current->next**.

Function to delete a node at given index or location from singly circular linked list:

```
// Function to delete node at given index
// of Circular Linked List
void DeleteAtPosition(struct Node** head, int index)
{
    // find length of list
    int len = Length(*head);
    int count = 1;
    struct Node *previous = *head, *next = *head;

    // check if list doesn't have any node
    // if not then return
    if (*head == NULL) {
        printf("\nDelete Last List is empty\n");
        return;
    }

    // given index is in list or not
    if (index >= len || index < 0) {
        printf("\nIndex is not Found\n");
        return;
    }

    // delete first node
    if (index == 0) {
        DeleteFirst(head);
        return;
    }

    // traverse first to last node
    while (len > 0) {

        // if index found delete that node
        if (index == count) {
            previous->next = next->next;
            free(next);
            return;
        }
        previous = previous->next;
        next = previous->next;
        len--;
        count++;
    }

    return;
}
```

Program implementing all of the above three functions

```
// C++ program to delete node at different
// positions from a circular linked list
#include <bits/stdc++.h>
using namespace std;

// structure for a node
struct Node {
    int data;
    struct Node* next;
};

// Function to insert a node at the end of
// a Circular linked list
void Insert(struct Node** head, int data)
{
    struct Node* current = *head;
    // Create a new node
    struct Node* newNode = new Node;

    // check node is created or not
    if (!newNode) {
        printf("\nMemory Error\n");
        return;
    }

    // insert data into newly created node
    newNode->data = data;

    // check list is empty
    // if not have any node then
    // make first node it
    if (*head == NULL) {
        newNode->next = newNode;
        *head = newNode;
        return;
    }

    // if list have already some node
    else {

        // move first node to last node
        while (current->next != *head) {
            current = current->next;
        }

        // put first or head node address
        // in new node link
        newNode->next = *head;

        // put new node address into last
        // node link(next)
        current->next = newNode;
    }
}

// Function print data of list
void Display(struct Node* head)
{
    struct Node* current = head;
    // if list is empty, simply show message
    if (head == NULL) {
        printf("\nDisplay List is empty\n");
        return;
    }

    // traverse first to last node
    else {
        do {
            printf("%d ", current->data);
            current = current->next;
        } while (current != head);
    }
}

// Function return number of nodes present in list
int Length(struct Node* head)
{
    struct Node* current = head;
    int count = 0;

    // if list is empty simply return length zero
    if (head == NULL) {
        return 0;
    }

    // traverse first to last node
    else {
        do {
            current = current->next;
            count++;
        } while (current != head);
    }
    return count;
}

// Function delete First node of Circular Linked List
```

```
void DeleteFirst(struct Node** head)
{
    struct Node *previous = *head, *next = *head;

    // check list have any node
    // if not then return
    if (*head == NULL) {
        printf("\nList is empty\n");
        return;
    }

    // check list have single node
    // if yes then delete it and return
    if (previous->next == previous) {
        *head = NULL;
        return;
    }

    // traverse second to first
    while (previous->next != *head) {

        previous = previous->next;
        next = previous->next;
    }

    // now previous is last node and
    // next is first node of list
    // first node(next) link address
    // put in last node(previous) link
    previous->next = next->next;

    // make second node as head node
    *head = previous->next;
    free(next);

    return;
}
```

```c
// Function to delete last node of
// Circular Linked List
void DeleteLast(struct Node** head)
{
    struct Node *current = *head, *temp = *head, *previous;

    // check if list doesn't have any node
    // if not then return
    if (*head == NULL) {
        printf("\nList is empty\n");
        return;
    }

    // check if list have single node
    // if yes then delete it and return
    if (current->next == current) {
        *head = NULL;
        return;
    }

    // move first node to last
    // previous
    while (current->next != *head) {
        previous = current;
        current = current->next;
    }

    previous->next = current->next;
    *head = previous->next;
    free(current);

    return;
}

// Function delete node at a given position
// of Circular Linked List
void DeleteAtPosition(struct Node** head, int index)
{
    // Find length of list
    int len = Length(*head);
    int count = 1;
    struct Node *previous = *head, *next = *head;

    // check list have any node
    // if not then return
    if (*head == NULL) {
        printf("\nDelete Last List is empty\n");
        return;
    }

    // given index is in list or not
    if (index >= len || index < 0) {
        printf("\nIndex is not Found\n");
        return;
    }

    // delete first node
    if (index == 0) {
        DeleteFirst(head);
        return;
    }

    // traverse first to last node
    while (len > 0) {

        // if index found delete that node
        if (index == count) {
            previous->next = next->next;
            free(next);
            return;
        }
        previous = previous->next;
        next = previous->next;
        len--;
        count++;
    }
    return;
}

// Driver Code
int main()
{
    struct Node* head = NULL;
    Insert(&head, 99);
    Insert(&head, 11);
    Insert(&head, 22);
    Insert(&head, 33);
    Insert(&head, 44);
    Insert(&head, 55);
    Insert(&head, 66);

    // Deleting Node at position
    printf("Initial List: ");
    Display(head);
    printf("\nAfter Deleting node at index 4: ");
    DeleteAtPosition(&head, 4);
    Display(head);

    // Deleting first Node
    printf("\n\nInitial List: ");
    Display(head);
    printf("\nAfter Deleting first node: ");
    DeleteFirst(&head);
    Display(head);

    // Deleting last Node
    printf("\n\nInitial List: ");
    Display(head);
    printf("\nAfter Deleting last node: ");
    DeleteLast(&head);
    Display(head);

    return 0;
}
```

Output:

```
Initial List: 99 11 22 33 44 55 66
After Deleting node at index 4: 99 11 22 33 55 66

Initial List: 99 11 22 33 55 66
After Deleting first node: 11 22 33 55 66

Initial List: 11 22 33 55 66
After Deleting last node: 11 22 33 55
```