Aryan Nanda **221070003**

EXPERIMENT 5

AIM : Removal of nodes from a Circular Linked List at a given interval(Johnson's Counter).

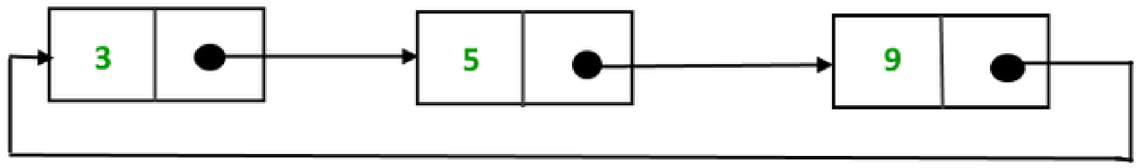
THEORY :

The circular linked list is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last node are connected to each other which forms a circle. There is no NULL at the end. There are generally two types of circular linked lists:

Circular singly linked list : In a circular Singly linked list, the last node of the list contains a pointer to the first node of the list. We traverse the circular singly linked list until we reach the same node where we started. The circular singly linked list has no beginning or end. No null value is present in the next part of any of the nodes.

Circular Doubly linked list : Circular Doubly Linked List has properties of both doubly linked list and circular linked list in which two consecutive elements are linked or connected by the previous and next pointer and the last node points to the first node by the next pointer and also the first node points to the last node by the previous pointer.

Node representation of a Circular Linked List:



Aryan Nanda **221070003**

<u>ALGORITHM</u> :

Absolutely, here's an algorithm in an exam-style format for the provided code that manipulates a circular linked list:

Algorithm: Circular Linked List Operations
Class: Node

Properties:

- data: Integer data stored in the node.
- next: Pointer to the next node.

Function: takeInput()

Read data until -1 is encountered from the input.

Create a circular linked list.

Return the head of the circular linked list.

Function: countNodes(head)

Initialize a pointer temp to the head.

If head is not NULL:

- Traverse the circular linked list using a do-while loop and increment a counter.
- Stop when temp reaches head again.

Return the count of nodes.

Function: printList(head)

If head is NULL, return.

Initialize a pointer temp to the head.

Traverse the circular linked list using a do-while loop:

- Print the data stored in temp.
- Move temp to the next node.
- Continue until temp reaches head again.

Function: freeList(node)

Calculate the total number of nodes in the circular linked list.

While the count of nodes is not zero:

- Print the data of the current node.
- Move to the next node.
- Delete the current node.
- Decrement the count of nodes.

Function: deleteEveryNth(head, n)

If head is NULL or n is less than or equal to 0, return.

Initialize pointers curr and prev to NULL.

If n is 1, free the entire list using the freeList() function.

Traverse the circular linked list:

- Move curr to the next node for n-1 times.
- Update the links to skip the nth node.

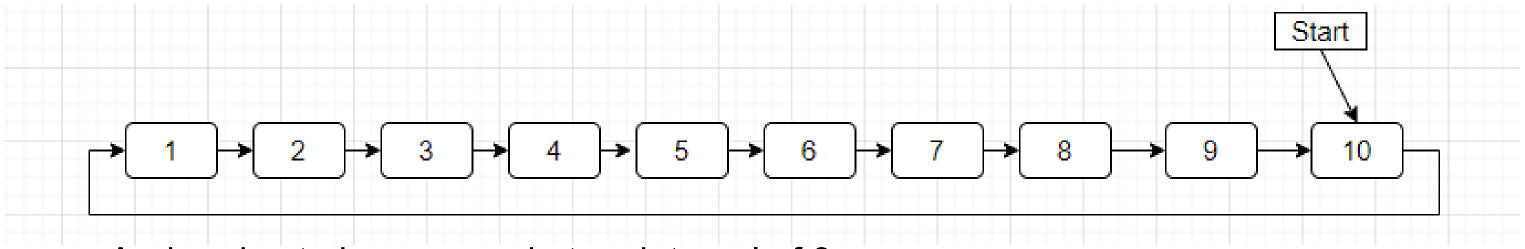
- Delete the nth node.
- Update pointers and handle circularity accordingly.

Delete the last remaining node.

Print the data of the last nod

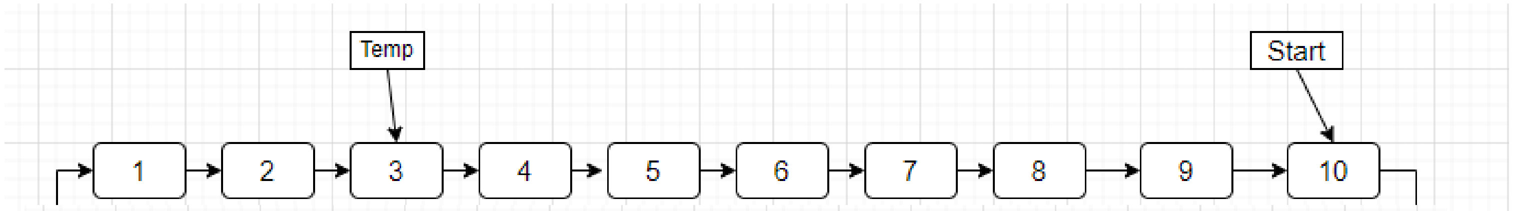
<u>EXAMPLE</u> :

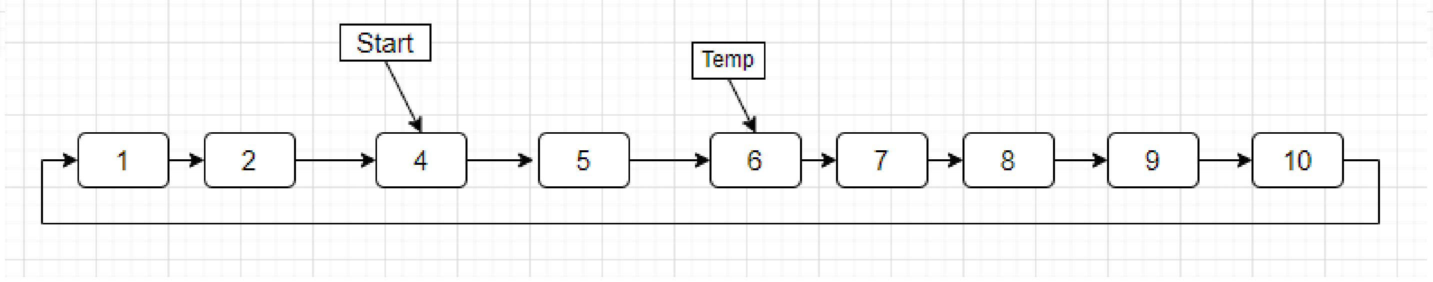
Consider the Circular Linked List -



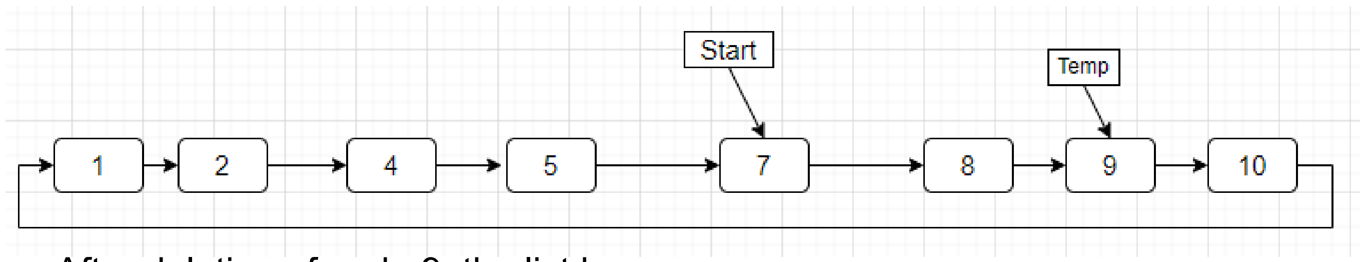
And nodes to be removed at an interval of 3.

The temp is set to 3 initially

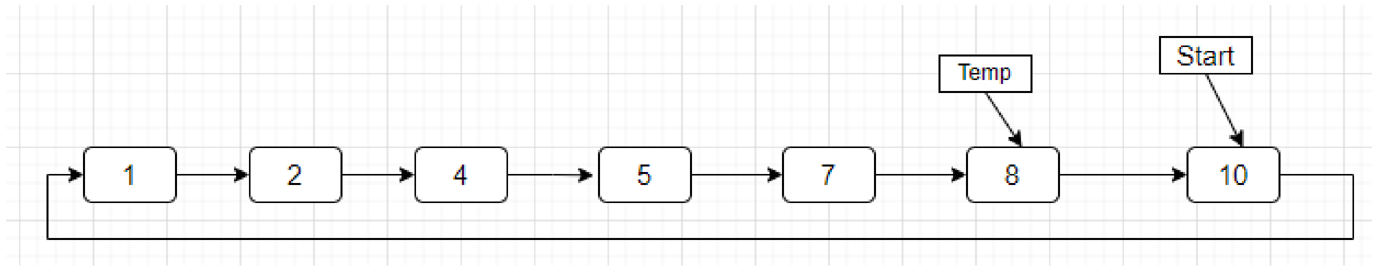




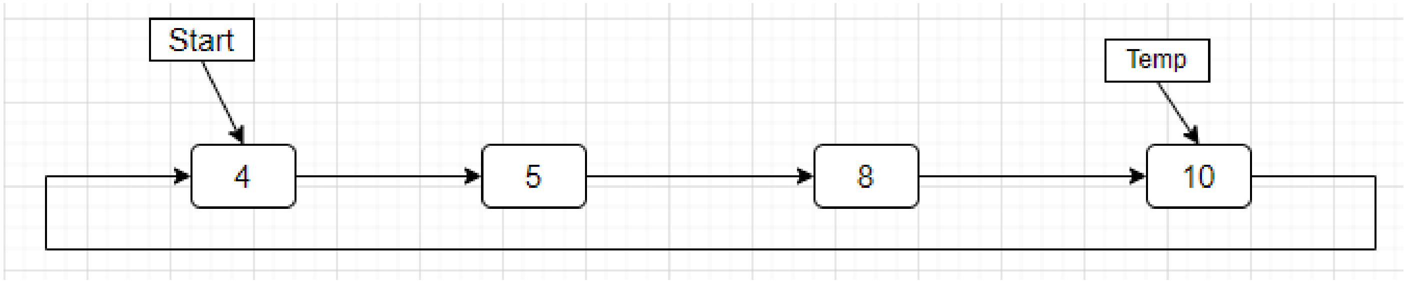
And then, 6 is deleted and temp is equal to node 9, start is 7



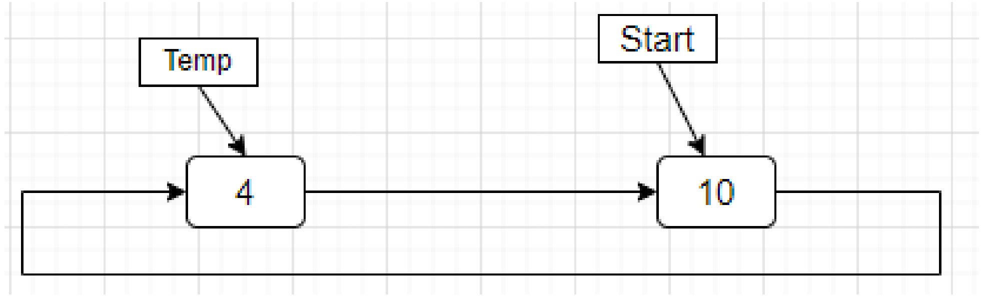
After deletion of node 9, the list becomes :



Similarly, in successive iteration, nodes 2, 7 and 1 are removed



Then, the nodes 8 and 5 are removed from the list



In the end, the nodes 10 and 4 are removed respectively.

CODE :

```
#include <iostream>
using namespace std;

class Node {
public:
   int data;
   Node* next;
   Node(int data) {
       this->data = data;
       this->next = nullptr;
   }
};

Node* takeInput() {
   int data;
   cin >> data;
   Node* head = nullptr;
```

```
    Node* tail = nullptr;

    while (data != -1) {
        Node* newNode = new Node(data);
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
        cin >> data;
    }

    if (tail != nullptr) {
        tail->next = head;
    }
    return head;
}
int countNodes(Node* head)
{
    Node* temp = head;
    int result = 0;
    if (head != NULL) {
        do {
            temp = temp->next;
            result++;
        } while (temp != head);
    }

    return result;
}
void printList(Node* head) {
    if (head == nullptr) return;

    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
}
void freeList(Node *node)
```

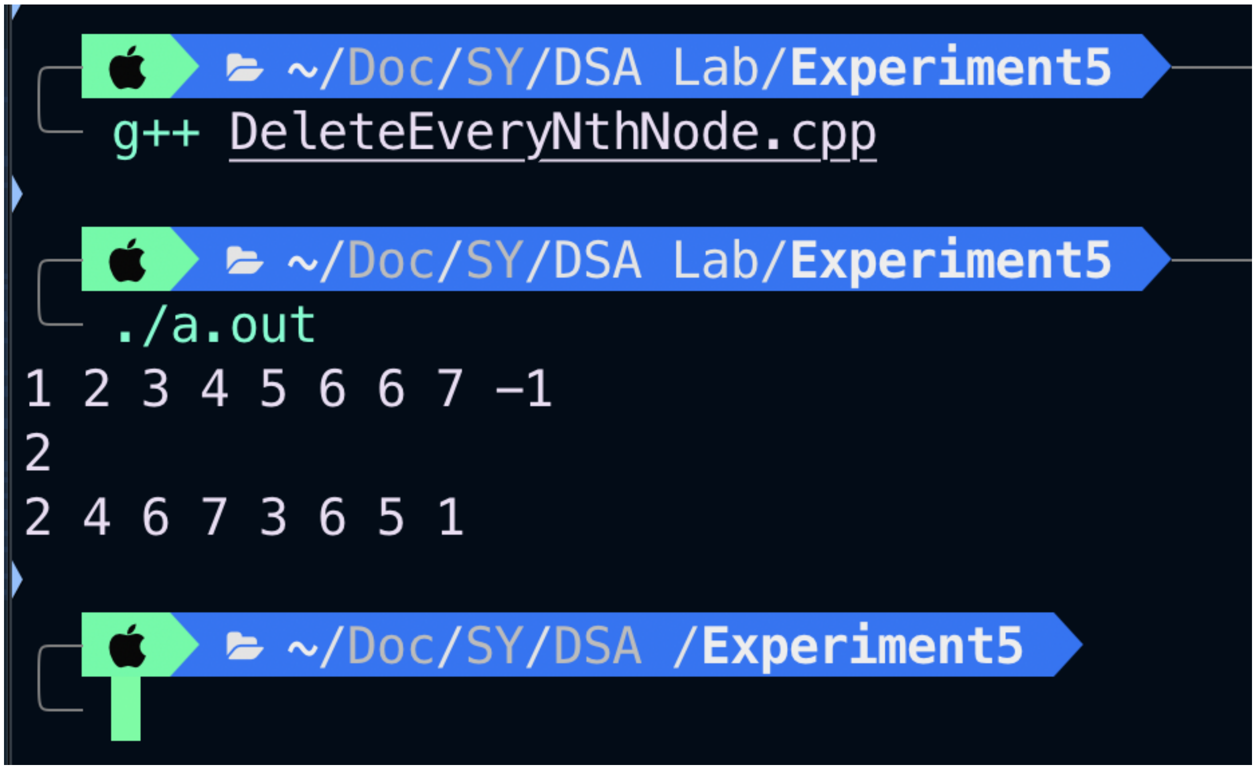
```
{
   int a = countNodes(node);
   while (a--)
   {
       cout<<node ->data<<" ";
       Node *next = node->next;
       delete (node);
       node  = next;
   }
}
void deleteEveryNth(Node* head, int n) {
   if (head == nullptr || n <= 0) return;
   Node* curr = head;
   Node* prev = nullptr;
   if(n==1){
       freeList(head);
       return;
   }
   while (curr->next != curr) {
       for (int i = 0; i < n - 1; ++i) {
           prev = curr;
           curr = curr->next;
       }

       prev->next = curr->next;
       Node* temp = curr;
       curr = curr->next;
       temp->next = nullptr;
       cout << temp->data << " ";
       delete temp;
       if (curr == head) {
           head = prev->next;
       }
   }
   cout << curr->data << endl;
   delete curr;
}

int main() {
   Node* head = takeInput();
   int n;
   cin>>n;
```

```
    deleteEveryNth(head, n);
    return 0;
}
```

Input and Output:-



```
~/Doc/SY/DSA Lab/Experiment5
g++ DeleteEveryNthNode.cpp

~/Doc/SY/DSA Lab/Experiment5
./a.out
1 2 3 4 5 6 6 7 -1
2
2 4 6 7 3 6 5 1

~/Doc/SY/DSA /Experiment5
```

<u>CONCLUSION</u> :

The experiment gave an insight into the properties of Circular Linked List and implemented a program to remove n^{th} node from the list until it is empty.