Data Structures and Algorithms

Department of Computer Science

University of Engineering and Technology, Lahore

CLOS: CLO1, CLO3

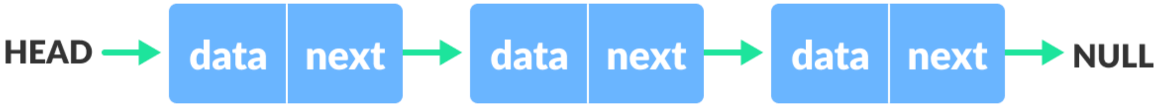
Singly Linked List

A **singly linked list** is a data structure in which each element (called a **node**) contains:

1. **Data** : the value or information stored.
2. **Pointer (next)** : the address (link) to the **next node** in the list.

The **last node's next pointer** is usually NULL, which means the list ends there.

Example of a normal singly linked list:



Circular Linked List:

Circular linked list is a linked list where all nodes are connected to form a circle. There is no NULL at the end. A circular linked list can be a singly circular linked list or doubly circular linked list.

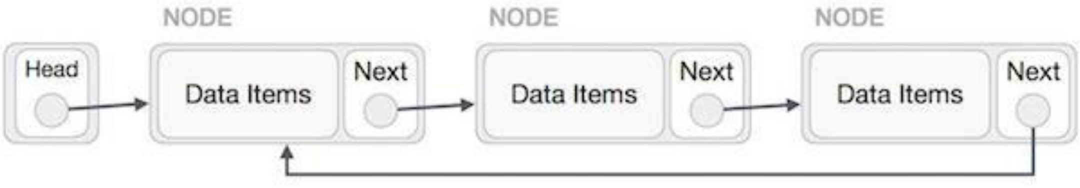


Figure 1. Singly List as Circular

- The last link's next points to the first link of the list in both cases of singly as well as doubly linked list.
- You can start from any node and eventually reach back to it.
- The traversal does not stop automatically; we must **manually check** if we've reached the starting point again.

Key Features:

1. No NULL at the end; it loops back to the start.
2. Can be traversed infinitely unless stopped manually.
3. Each node has one link (just like singly linked list).

4. Efficient for circular tasks like **round-robin scheduling** or **playing songs in a music playlist loop**.

Step 1: Define the Node Structure

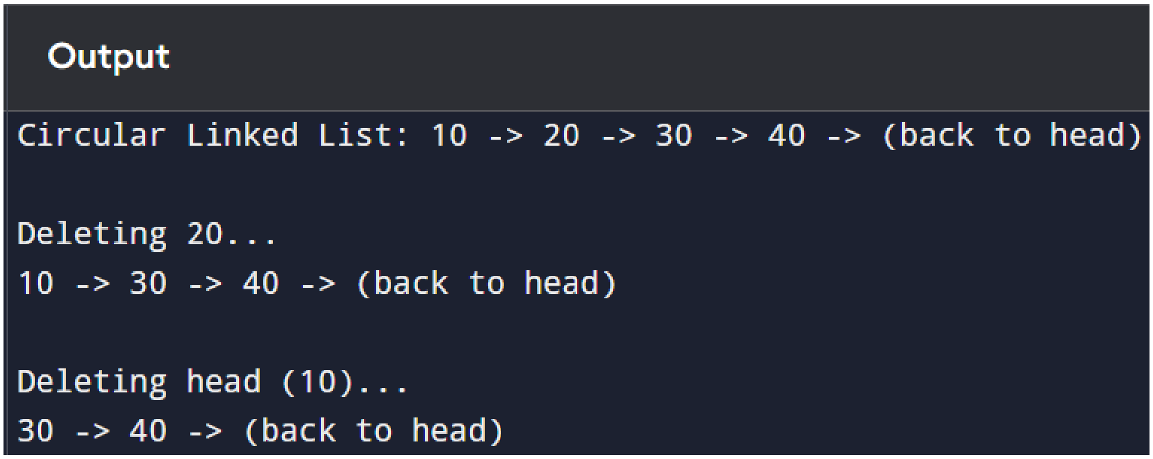
```
#include <iostream>
using namespace std;
class Node {
public:
    int data;
    Node* next;
    Node(int val) {
        data = val;
        next = NULL;
    }
};
```

Step 2: Create a Class for Circular Singly Linked List

```
class CircularLinkedList {
public:
    Node* head;
    CircularLinkedList() {
        head = NULL;
    }
    // Function to insert node at end
    void insertEnd(int val) {
        Node* newNode = new Node(val);
        if (head == NULL) {
            head = newNode;
            newNode->next = head; // points to itself
            return;
        }
        Node* temp = head;
        while (temp->next != head) { // find last node
        temp = temp->next;
    }
    temp->next = newNode;
    newNode->next = head; // make it circular again
}
// Function to display the list
void display() {
    if (head == NULL) {
        cout << "List is empty\n";
        return;
    }
    Node* temp = head;
    do {
        cout << temp->data << " -> ";
        temp = temp->next;
    } while (temp != head);
    cout << "(back to head)" << endl;
}
// Function to delete a node by value
void deleteNode(int val) {
    if (head == NULL) {
        cout << "List is empty!\n";
        return;
    }
    Node* current = head;
    Node* previous = NULL;
  // Case 1: Only one node in the list
    if (head->next == head && head->data == val) {
        delete head;
        head = NULL;
        return;
        }
        // Case 2: Delete the head node
        if (head->data == val) {
            Node* temp = head;
            while (temp->next != head) {
                temp = temp->next;
            }
            temp->next = head->next;
            Node* del = head;
            head = head->next;
            delete del;
            return;
        }
        // Case 3: Delete non-head node
        do {
            previous = current;
            current = current->next;
            if (current->data == val) {
                previous->next = current->next;
                delete current;
                return;
            }
        } while (current != head);
        cout << "Value not found!\n";
    }
};
```

Step 3: Main Function

```
int main() {
    CircularLinkedList cll;
    cll.insertEnd(10);
    cll.insertEnd(20);
    cll.insertEnd(30);
    cll.insertEnd(40);
    cout << "Circular Linked List: ";
    cll.display();
    cout << "\nDeleting 20...\n";
    cll.deleteNode(20);
    cll.display();
    cout << "\nDeleting head (10)...\n";
    cll.deleteNode(10);
    cll.display();
    return 0;
}
```



Output

```
Circular Linked List: 10 -> 20 -> 30 -> 40 -> (back to head)

Deleting 20...
10 -> 30 -> 40 -> (back to head)

Deleting head (10)...
30 -> 40 -> (back to head)
```

Practice Questions

Q1. Insert a Node After Every Kth Node

Given a circular singly linked list and an integer **k**, insert a new node with value **X** after every kth node.

Example:

Input: 10 → 20 → 30 → 40 → 50 → (back to 10), **k = 2, X = 99**
Output: 10 → 20 → 99 → 30 → 40 → 99 → 50 → (back to 10)

Q2. Delete Every Mth Node Until One Node Remains (Josephus Problem)

You are given a circular singly linked list of **n** people standing in a circle. Starting from the head, count **m** nodes and delete the **mth** node each time. Continue until only one node remains. Return the data of that last node.

Example:

Input: n = 5, m = 2

List: 1 → 2 → 3 → 4 → 5 → (back to 1)

Output: 3 (3 survives)

Hint: This is the famous Josephus Problem. Use circular list traversal and repeatedly delete every **mth** node.

Q3. Split Circular List into Odd and Even Nodes

Problem: Split a circular singly linked list into two separate circular lists —
one containing node at odd positions, the other containing nodes at even positions.

Example:

Input: 10 → 20 → 30 → 40 → 50 → 60 → (back to 10)

Output:

Odd List: 10 → 30 → 50 → (back to 10)

Even List: 20 → 40 → 60 → (back to 20)

Hint: Maintain two circular lists (oddHead, evenHead); alternate nodes while traversing the main list.