Introduction to Circular Linked List

A circular linked list is a data structure where the last node points back to the first node, forming a closed loop.

- **Structure:** All nodes are connected in a circle, enabling continuous traversal without encountering **`NULL`.**

- Difference from Regular Linked List: In a regular linked list, the last node points to `NULL`, whereas in a circular linked list, it points to the first node.

- **Uses:** Ideal for tasks like scheduling and managing playlists, where smooth and repeated.

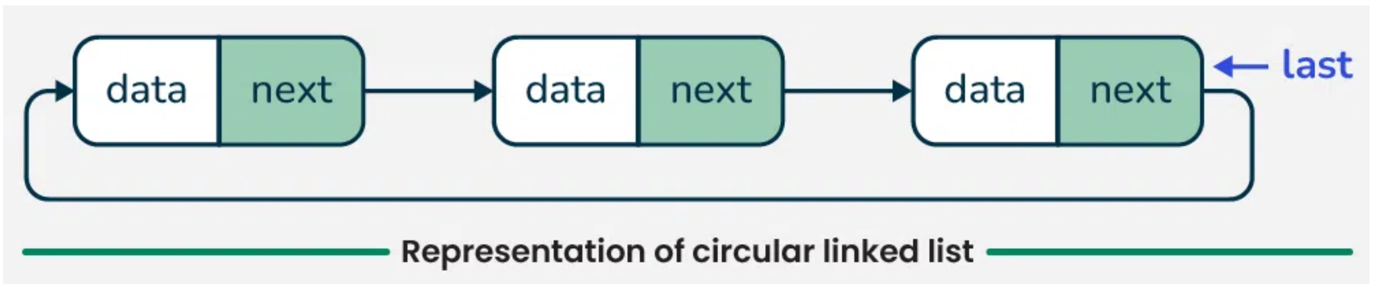
Types of Circular Linked Lists

We can create a circular linked list from both
singly linked lists and
 doubly linked lists.

1. Circular Singly Linked List

In Circular Singly Linked List, each node has just one pointer called the "next" pointer. The next pointer of the last node points back to the first node and this results in forming a circle. In this type of Linked list, we can only move through the list in one direction.

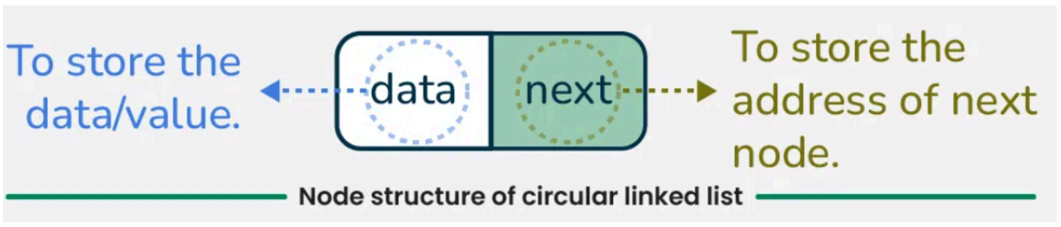
Representation of Circular Singly Linked List



```
Note: Here, we will use the singly linked list to explain the
working of circular linked lists.
```

Representation of a Circular Singly Linked List

Let's take a look on the structure of a circular linked list.


To store the data/value.
data
next
To store the address of next node.
Node structure of circular linked list
Representation of a Circular Singly Linked List

Representation of a Circular Singly Linked List

Create/Declare a Node of Circular Linked List

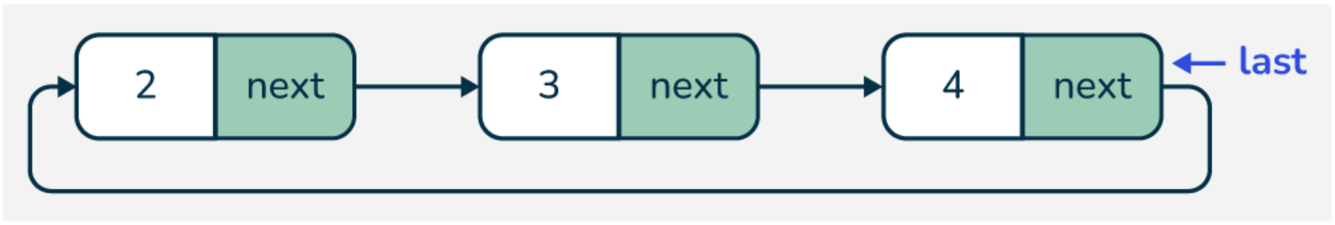
Syntax to Declare a Circular Linked List in C Language:

```c
// Node structure
struct Node
{
    int data;
    struct Node *next;
};

// Function to create a new node
struct Node *createNode(int value){

    // Allocate memory
    struct Node *newNode =
      (struct Node *)malloc(sizeof(struct Node));

    // Set the data
    newNode->data = value;

    // Initialize next to NULL
    newNode->next = NULL;

    // Return the new node
    return newNode;
}
```

In the code above, each node has **data** and a **pointer** to the next node. When we create multiple nodes for a circular linked list, we only need to connect the last node back to the first one.

Example of Creating a Circular Linked List

Here's an example of creating a circular linked list with three nodes (2, 3, 4):



Created a circular linked list with 3 nodes

```c
// Allocate memory for nodes
struct Node *first =
  (struct Node *)malloc(sizeof(struct Node));
struct Node *second =
  (struct Node *)malloc(sizeof(struct Node));
struct Node *last =
  (struct Node *)malloc(sizeof(struct Node));

// Initilize nodes
first->data = 2;
second->data = 3;
last->data = 4;

// Connect nodes
first->next = second;
second->next = last;
last->next = first;
```

In the above code, we have created three nodes first, second, and last having values 2, 3, and 4 respectively.

- After creating three nodes, we have connected these nodes in a series.
- Connect the first node "first" to "second" node by storing the address of "second" node into first's next
- Connect the second node "second" to "third" node by storing the address of "third" node into second's next
- After connecting all the nodes, we reach the key characteristic of a circular linked list: linking the last node back to the first node. Therefore, we store the address of the "first" node in the "last" node.

Why have we taken a pointer that points to the last node instead of the first node?

For the insertion of a node at the beginning, we need to traverse the whole list. Also, for insertion at the end, the whole list has to be traversed. If instead of the start pointer, we take a pointer to the last node, then in both cases there won't be any need to traverse the whole list. So insertion at the beginning or at the end takes constant time, irrespective of the length of the list.

Operations on the Circular Linked list

1. Create
2. Delete

Program to Implement Circular Linked List in C

```c
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct Node {
    int data;
    struct Node *next;
};

struct Node *head = NULL;   // Head pointer

// Function to create/insert a node at the end
void create(int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;

    if (head == NULL) {
        head = newNode;
        head->next = head;   // Point to itself
    } else {
        struct Node *temp = head;
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;  // Complete the circle
    }
    printf("%d inserted.\n", value);
}

// Function to delete a node by value
void deleteNode(int value) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node *current = head, *prev = NULL;

    // Case 1: Only one node
    if (head->data == value && head->next == head) {
        free(head);
        head = NULL;
        printf("%d deleted.\n", value);
        return;
    }
    // Case 2: Deleting head node
    if (head->data == value) {
        struct Node *last = head;
        while (last->next != head) {
            last = last->next;
        }
        last->next = head->next;
        struct Node *temp = head;
        head = head->next;
        free(temp);
        printf("%d deleted.\n", value);
        return;
    }

    // Case 3: Deleting other nodes
    prev = head;
    current = head->next;
    while (current != head) {
        if (current->data == value) {
            prev->next = current->next;
            free(current);
            printf("%d deleted.\n", value);
            return;
        }
        prev = current;
        current = current->next;
    }

    printf("Node with value %d not found.\n", value);
}

// Function to display list
void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node *temp = head;
    printf("Singly Circular Linked List: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to head)\n");
}

// Main menu
```

```
int main() {
    int choice, value;

    while (1) {
        printf("\n--- Singly Circular Linked List Menu ---\n");
        printf("1. Create (Insert at End)\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                create(value);
                break;
            case 2:
                printf("Enter value to delete: ");
                scanf("%d", &value);
                deleteNode(value);
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }

    return 0;
}
```

Applications of Circular Linked List

1. **Round Robin Scheduling:** It can be used in the operating system to schedule a process in the cyclic manner.
2. **FIFO Buffers**: It can be used in networking for the buffer management.
3. **Music Player Playlists**: It can be used to cycle through the songs in a playlist.