**NAME: SUVIDHI V. PAREEK  
SE(COMPS) / DIV-3 / ROLL NO. 10**

**Experiment no 7:**

**Aim: Implementation of Circular Linked List ADT**

**Objective: Circular Linked List can be used to manage the Computing Resources of the computer. Data Structure such as stacks and queue are implemented with the help of circular linked list**

**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.***

**Algorithm:**

**Insertion on a Circular Linked List:**

### **1. Insertion at the Beginning**

* **store the address of the current first node in the newNode (i.e. pointing the newNode to the current first node)**
* **point the last node to newNode (i.e making newNode as head)**

### **2. Insertion in between two nodes**

**Let's insert newNode after the first node.**

* **travel to the node given (let this node be p)**
* **point the next of newNode to the node next to p**
* **store the address of newNode at next of p**

### 

### **3. Insertion at the end**

* **store the address of the head node to next of newNode (making newNode the last node)**
* **point the current last node to newNode**
* **make newNode as the last node**

### **Deletion on a Circular Linked List**

### **1. If the node to be deleted is the only node**

* **free the memory occupied by the node**
* **store NULL in last**

### **2. If last node is to be deleted**

* **find the node before the last node (let it be temp)**
* **store the address of the node next to the last node in temp**
* **free the memory of last**
* **make temp as the last node**

### **3. If any other nodes are to be deleted**

**Suppose we have a circular linked list with elements 1, 2, and 3.**

* **travel to the node to be deleted (here we are deleting node 2)**
* **let the node before node 2 be temp**
* **store the address of the node next to 2 in temp**
* **free the memory of 2**

**Code:**

**#include <stdio.h>**

**#include <stdlib.h>**

**// Structure for a node**

**struct Node {**

**int data;**

**struct Node\* next;**

**};**

**// Function to insert a node at the**

**// beginning of a Circular linked list**

**void push(struct Node\*\* head\_ref, int data)**

**{**

**// Create a new node and make head**

**// as next of it.**

**struct Node\* ptr1 = (struct Node\*)malloc(sizeof(struct Node));**

**ptr1->data = data;**

**ptr1->next = \*head\_ref;**

**// If linked list is not NULL then**

**// set the next of last node**

**if (\*head\_ref != NULL) {**

**// Find the node before head and**

**// update next of it.**

**struct Node\* temp = \*head\_ref;**

**while (temp->next != \*head\_ref)**

**temp = temp->next;**

**temp->next = ptr1;**

**}**

**else**

**// For the first node**

**ptr1->next = ptr1;**

**\*head\_ref = ptr1;**

**}**

**// Function to print nodes in a given**

**// circular linked list**

**void printList(struct Node\* head)**

**{**

**struct Node\* temp = head;**

**if (head != NULL) {**

**do {**

**printf("%d ", temp->data);**

**temp = temp->next;**

**} while (temp != head);**

**}**

**printf("\n");**

**}**

**// Function to delete a given node**

**// from the list**

**void deleteNode(struct Node\*\* head, int key)**

**{**

**// If linked list is empty**

**if (\*head == NULL)**

**return;**

**// If the list contains only a**

**// single node**

**if ((\*head)->data == key && (\*head)->next == \*head) {**

**free(\*head);**

**\*head = NULL;**

**return;**

**}**

**struct Node \*last = \*head, \*d;**

**// If head is to be deleted**

**if ((\*head)->data == key) {**

**// Find the last node of the list**

**while (last->next != \*head)**

**last = last->next;**

**// Point last node to the next of**

**// head i.e. the second node**

**// of the list**

**last->next = (\*head)->next;**

**free(\*head);**

**\*head = last->next;**

**return;**

**}**

**// Either the node to be deleted is**

**// not found or the end of list**

**// is not reached**

**while (last->next != \*head && last->next->data != key) {**

**last = last->next;**

**}**

**// If node to be deleted was found**

**if (last->next->data == key) {**

**d = last->next;**

**last->next = d->next;**

**free(d);**

**}**

**else**

**printf("Given node is not found in the list!!!\n");**

**}**

**// Driver code**

**int main()**

**{**

**// Initialize lists as empty**

**struct Node\* head = NULL;**

**// Created linked list will be**

**// 2->5->7->8->10**

**push(&head, 2);**

**push(&head, 5);**

**push(&head, 7);**

**push(&head, 8);**

**push(&head, 10);**

**printf("List Before Deletion: ");**

**printList(head);**

**deleteNode(&head, 7);**

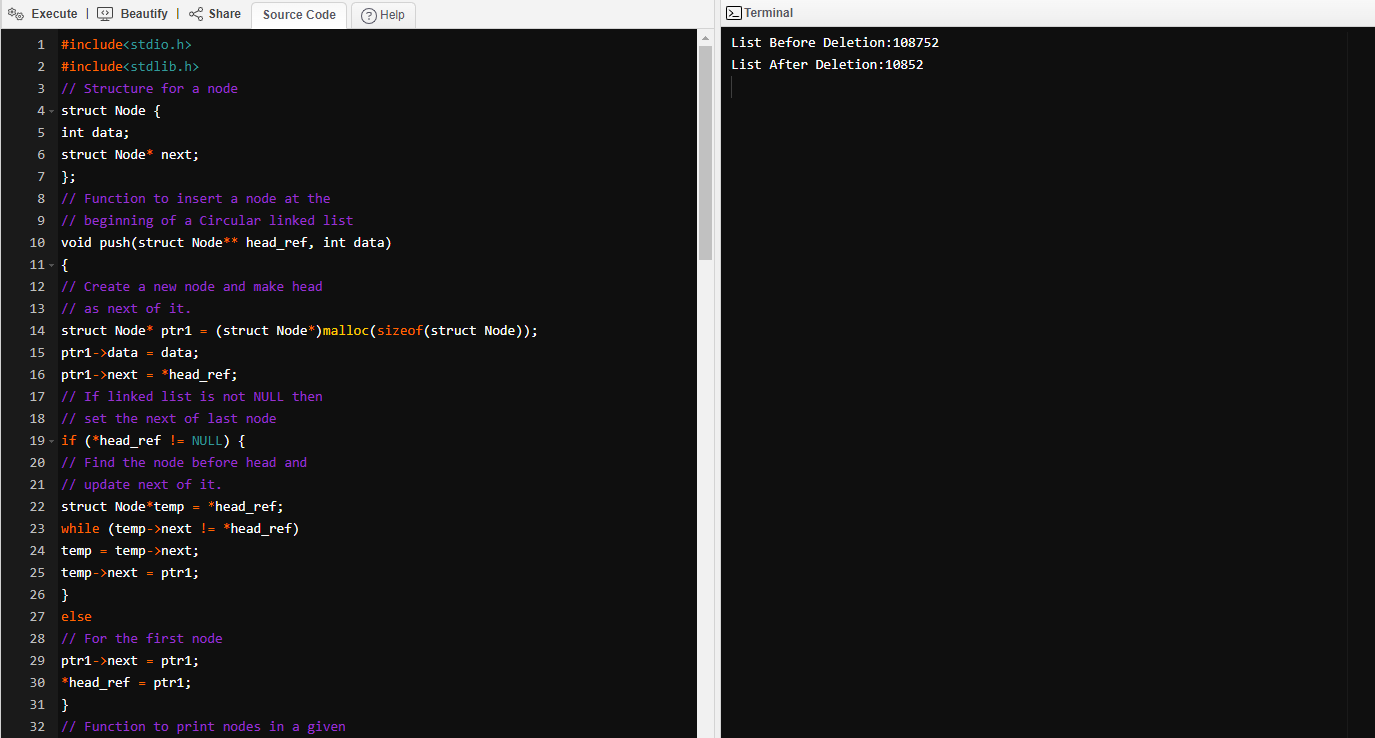
**printf("List After Deletion: ");**

**printList(head);**

**return 0;**

**}**

**Output:**

****

**Conclusion :**

1. **The NULL assignment is not required because a node always points to another node.**
2. **The starting point can be set to any node.**
3. **Traversal from the first node to the last node is quick.**