Aim: Implement Circular Linked List ADT.

Theory:

A circular linked list is a type of linked list where the last node of the list points back to the first node instead of pointing to NULL. This circular linking makes traversal of the list from any point possible and eliminates the need to maintain a separate pointer for the head and tail of the list.

last node pointer back to first node, creating the circular link. This organization allows for the seamless transitions from the end of list back to the beginning and it can be enabling the continuous iteration over the lists elements.

Procedure:

1 **Define Node Structure**:

- Create a structure to represent a node.
- Include a data field (e.g., an integer) and a pointer to the next node.

2 **Create Node Function**:

- Write a function to create a new node.
- Initialize the data field and set the next pointer to point to itself.

3**Insert Node Function**:

- Write a function to insert a node at the end of the circular linked list.
- If the list is empty, set the head to the new node.
- If the list is not empty:
    - Traverse to the last node (where its next pointer points to the head). ○ Update the last node's next pointer to the new node.
    - Set the new node's next pointer to the head.

4 **Delete Node Function**:

- Write a function to delete a node with a specific value (key).
- Handle the case where the node to be deleted is the head:
        - If the list has only one node, free it and set the head to NULL.

- If there are multiple nodes, find the last node and update its next pointer.

- For non-head nodes, traverse the list to find the node to be deleted and update the pointers accordingly.

5 **Display List Function**:

- Write a function to display the elements of the circular linked list.
- Start from the head and traverse the list until you return to the head, printing each node's data.

6 **Main Function**:

- Create the main function to test the circular linked list implementation.
- Initialize the head of the list.
- Use the insertion function to add nodes.
- Display the list to confirm nodes are added correctly.
- Use the deletion function to remove nodes and display the list after each deletion.

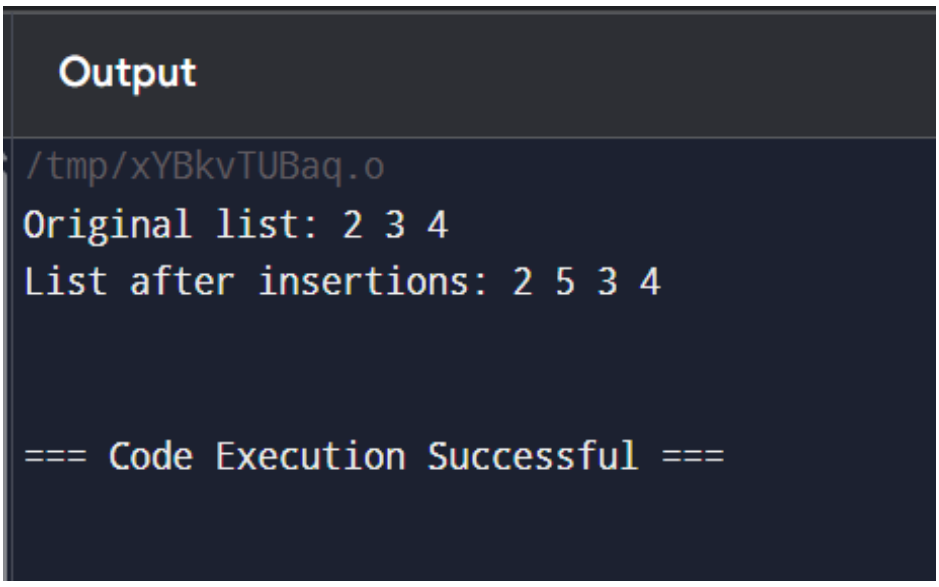
Code:

```
#include <stdio.h>
 #include <stdlib.h>
 // Define the Node structure
 struct Node {
 int data;
struct Node *next;
 };
 struct Node* createNode(int value);
 // Function to insert a node at a specific position in a circular linked list
 struct Node* insertAtPosition(struct Node *last, int data, int pos) {
 if (last == NULL) {
 // If the list is empty
 if (pos != 1) {
 printf("Invalid position!\n");
 return last;
 }
 // Create a new node and make it point to itself
```

```
 struct Node *newNode = createNode(data);
 last = newNode;
 last->next = last;
 return last;
 }
// Create a new node with the given data
 struct Node *newNode = createNode(data);
 // curr will point to head initially
 struct Node *curr = last->next;
 if (pos == 1) {
 // Insert at the beginning
 newNode->next = curr;
 last->next = newNode;
 return last;
 }
 // Traverse the list to find the insertion point
 for (int i = 1; i < pos- 1; ++i) {
 curr = curr->next;
 // If position is out of bounds
 if (curr == last->next) {
printf("Invalid position!\n");
 return last;
 }
 }
 // Insert the new node at the desired position
 newNode->next = curr->next;
 curr->next = newNode;
 // Update last if the new node is inserted at the end
 if (curr == last) last = newNode;
 return last;
 }
 // Function to print the circular linked list
 void printList(struct Node *last) {
 if (last == NULL) return;
 struct Node *head = last->next;
while (1) {
 printf("%d ", head->data);
 head = head->next;
 if (head == last->next) break;
 }
 printf("\n");
```

```
}
// Function to create a new node
struct Node* createNode(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
return newNode;
}
int main() {
// Create circular linked list: 2, 3, 4
struct Node *first = createNode(2);
first->next = createNode(3);
first->next->next = createNode(4);
struct Node *last = first->next->next;
last->next = first;
printf("Original list: ");
printList(last);
// Insert elements at specific positions
int data = 5, pos = 2;
last = insertAtPosition(last, data, pos);
printf("List after insertions: ");
printList(last);
return 0;
}
```

Output:



Conclusion: This experiment builds upon the previous one by implementing a Circular Linked List ADT. It illustrates the use of circular linked lists, where the last node points back to the first node, enabling efficient traversal and avoiding null pointer issues.