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

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

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

// Function to insert a node at the end of the circular linked list
void insertNode(struct Node** head, int data) {
    struct Node* newNode = createNode(data);

    if (*head == NULL) {
        // If the list is empty, make the new node the head and point to itself
        *head = newNode;
        newNode->next = *head;
    } else {
        // Traverse to the last node
        struct Node* temp = *head;
        while (temp->next != *head) {
            temp = temp->next;
        }

        // Insert the new node at the end
        temp->next = newNode;
        newNode->next = *head;
    }
}

// Function to delete a node with a given data value from the circular linked list
void deleteNode(struct Node** head, int data) {
    if (*head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct Node *current = *head, *prev = NULL;
// If the node to be deleted is the head
    if (current->data == data) {
        prev = *head;
        while (prev->next != *head) {
            prev = prev->next;
        }

        if (*head == (*head)->next) {
            // If there is only one node in the list
            free(*head);
            *head = NULL;
        } else {
            // If there are multiple nodes in the list
            prev->next = current->next;
            *head = current->next;
            free(current);
        }
    } else {
        // Search for the node to be deleted
        while (current->next != *head && current->data != data) {
            prev = current;
            current = current->next;
        }

        if (current->data == data) {
            // Delete the node
            prev->next = current->next;
            free(current);
        } else {
            printf("Node with data %d not found\n", data);
        }
    }
}

// Function to traverse and print the circular linked list
void traverseList(struct Node* head) {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }

    struct Node* current = head;
// Traverse and print the circular linked list
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != head);

    printf("(head)\n");
}

// Function to free the memory allocated for the circular linked list
void freeList(struct Node** head) {
    if (*head == NULL) {
        return;
    }

    struct Node *current = *head, *temp;

    // Traverse and free each node
    do {
        temp = current;
        current = current->next;
        free(temp);
    } while (current != *head);

    *head = NULL;
}

int main() {
    struct Node* head = NULL;
    int choice, data;

    do {
        printf("\n1. Insert Node\n");
        printf("2. Delete Node\n");
        printf("3. Traverse List\n");
        printf("0. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: printf("Enter data for the new node: ");
                scanf("%d", &data);
                insertNode(&head, data);
                break;
            case 2:
                printf("Enter data of the node to delete: ");
                scanf("%d", &data);
                deleteNode(&head, data);
                break;
            case 3:
```

```
            printf("Circular Linked List:\n");
            traverseList(head);
            break;
        case 0:
            printf("Exiting the program.\n");
            break;
        default:
            printf("Invalid choice. Please enter a valid option.\n");
      }
   } while (choice != 0);

   // Free the memory allocated for the circular linked list
   freeList(&head);

   return 0;
}
```