```c
// Write a menu driven program in C to implement a Circular Linked List
using Dynamic Memory Allocation and apply
// various operations on it through separate functions as below:
// a)Create a Circular linked list
// b)Display the list of elements
// c)Insert a node at the beginning of the list
// d)Insert a node at the end of the list
// e)Insert a node at a given position in the list
// f)Insert a node after a given node
// g)Delete the first node
// h)Delete the last node
// i)Delete a node at a given position
// j)Delete a node after a given node
// k)Search an element in the list
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int val;
    struct node *next;
};
struct node *head;

void create();
void traverse();
void insertAtFront();
void insertAtEnd();
void insertAtPosition();
void insertNode();
void deleteFirst();
void deleteFirst();
void deleteEnd();
void deletePosition();
void deleteNode();
int search(struct node *x, int item);

int main()
{
    int choice = 0;
    while (choice != 7)
    {
        int i, key;
        printf("\n\t1 Create a circular linked list\n");
        printf("\t2 Display the list of elements\n");
        printf("\t3 Insert a node at the beginning of the list\n");
        printf("\t4 Insert a node at the end of the list\n");
        printf("\t5 Insert a node at a given position in the list\n");
        printf("\t6 Insert a node after a given node\n");
        printf("\t7 Delete the first node\n");
        printf("\t8 Delete the last node\n");
        printf("\t9 Delete a node at a given position\n");
        printf("\t10 Delete a node after a given node\n");
        printf("\t11 Search an element in the list\n");
        printf("\t11 Exit\n");

        printf("\nEnter your choice?\n");
        scanf("\n%d", &choice);

        switch (choice)
        {
        case 1:
            create();
            break;
        case 2:
            traverse();
            break;
        case 3:
            insertAtFront();
            break;
        case 4:
            insertAtEnd();
            break;
        case 5:
            insertAtPosition();
            break;
        case 6:
            insertNode();
            break;
        case 7:
            deleteFirst();
            break;
        case 8:
            deleteEnd();
            break;
        case 9:
            deletePosition();
            break;
        case 10:
            deleteNode();
            break;
        case 11:
            printf("Enter the key you want to search\n");
            scanf("%d",&key);
            i = search(head,key);
            printf("The key is found at %d index", i);
            break;
        case 12:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
}
void create()
{
    struct node *temp;
    temp = (struct node *)malloc(sizeof(struct node));
    head = temp;
    char ch;
    printf("Enter the value of node\n");
    scanf("%d", &temp->val);
    temp->next = head;
    printf("Enter any character to continue or q for quit\n");
    scanf(" %c", &ch);
    while (ch != 'q')
    {
        temp->next = (struct node *)malloc(sizeof(struct node));
        if (temp->next == NULL)
        {
            printf("Memory is not allocated\n");
            exit(0);
        }
        temp = temp->next;
        printf("Enter thr value of node\n");
        scanf("%d", &temp->val);
        temp->next = head;
        printf("Enter any character to continue or q for quit\n");
        scanf(" %c", &ch);
    }
}
void traverse()
{
    struct node *ptr;
    ptr = head;
    if (head == NULL)
    {
        printf("\nnothing to print");
    }
    else
    {
        printf("\n printing values ... \n");

        while (ptr->next != head)
        {

            printf("%d\t",ptr->val);
            ptr = ptr->next;
        }
        printf("%d\t", ptr->val);
    }
    printf("\n\n");
}
void insertAtFront()
{
    struct node *ptr, *temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW");
    }
    else
    {
        printf("\nEnter the node val?");
        scanf("%d", &item);
        ptr->val = item;
        if (head == NULL)
        {
            head = ptr;
            ptr->next = head;
        }
        else
        {
            temp = head;
            while (temp->next != head)
                temp = temp->next;
            ptr->next = head;
            temp->next = ptr;
            head = ptr;
        }
        printf("\nnode inserted\n");
    }
}
```

```
void insertAtEnd()
{
    struct node *ptr, *temp;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if (ptr == NULL)
    {
        printf("\nOVERFLOW\n");
    }
    else
    {
        printf("\nEnter Data?");
        scanf("%d", &item);
        ptr->val = item;
        if (head == NULL)
        {
            head = ptr;
            ptr->next = head;
        }
        else
        {
            temp = head;
            while (temp->next != head)
            {
                temp = temp->next;
            }
            temp->next = ptr;
            ptr->next = head;
        }

        printf("\nnode inserted\n");
    }
}

void insertAtPosition()
{
    struct node *temp, *newnode;
    int pos, data, i = 1;
    newnode = malloc(sizeof(struct node));

    // Enter the position and data
    printf("\nEnter position and data :");
    scanf("%d %d", &pos, &data);

    // Change Links
    temp = head;
    newnode->val = data;
    newnode->next = head;
    while (i < pos - 1)
    {
        temp = temp->next;
        i++;
    }
    newnode->next = temp->next;
    temp->next = newnode;
}
void insertNode()
{
    int key;
    printf("Enter the position after swhich you want to add a new
node\n");
    scanf("%d",&key);
    struct node *new,*temp;
    temp=head;
    int i,sz,n;
    n=search(temp, key);
    n++;
    for(sz=1; temp->next!=head; sz++)
        temp=temp->next;
    if(n>1 && n<=sz)
    {
        for(i=1; i<n-1; i++)
            temp=temp->next;
        new=(struct node *)malloc(sizeof(struct node));
        if(new==NULL)
        {
            printf("\nMemory not allocated Properly !\n");
            exit(0);
        }
        printf("\nEnter the Value : ");
        scanf("%d",&new->val);
        (new->next)=(temp->next);
        temp->next=new;
    }
    else if(n==sz+1)
        insertAtEnd();
    else
        printf("\nNode Not Found !\n");
}

void deleteFirst()
{
    struct node *ptr;
    if (head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }

    else
    {
        ptr = head;
        while (ptr->next != head)
            ptr = ptr->next;
        ptr->next = head->next;
        free(head);
        head = ptr->next;
        printf("\nnode deleted\n");
    }
}
void deleteEnd()
{
    struct node *ptr, *preptr;
    if (head == NULL)
    {
        printf("\nUNDERFLOW");
    }
    else if (head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        while (ptr->next != head)
        {
            preptr = ptr;
            ptr = ptr->next;
        }
        preptr->next = ptr->next;
        free(ptr);
```

```c
        printf("Node Deleted\n");
    }
}
void deletePosition()
{
    struct node *temp, *position;
    int i = 1, pos;

    // If LL is empty
    if (head == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else
    {
        printf("\nEnter index : ");

        // Position to be deleted
        scanf("%d", &pos);
        position = malloc(sizeof(struct node));
        temp = head;

        // Traverse till position
        while (i < pos - 1)
        {
            temp = temp->next;
            i++;
        }

        // Change Links
        position = temp->next;
        temp->next = position->next;

        // Free memory
        free(position);
    }
}
void deleteNode()
{
    int key;
    printf("Enter the position\n");
    scanf("%d",&key);
    struct node *temp;
    int n,sz;
    temp=head;
    for(sz=1; temp->next!=head; sz++)
```

```c
        temp=temp->next;
    n=search(temp,key);
    n++;
    if(n==sz)
        deleteEnd();
    else if(n>1 && n<sz)
    {
        int i;
        for(i=1; i<n; i++)
        {
            temp=head;
            head=head->next;
        }
        (temp->next)=(head->next);
        free(head);
    }
    else if(n==sz+1)
        printf("\nNo element is present !\n");
    else
        printf("\nNode is Absent !\n");
}

int search(struct node *x, int item)
{
    struct node *ptr;
    int i = 0, flag = 1;
    ptr = head;
    if (ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        if (head->val == item)
        {
            return i+1;
            flag = 0;
        }
        else
        {
            while (ptr->next != head)
            {
                if (ptr->val == item)
                {
                    return i + 1;
                    flag = 0;
                    break;
                }
                else
                {
                    flag = 1;
                }
                i++;
                ptr = ptr->next;
            }
        }
        if (flag != 0)
        {
            printf("Item not found\n");
        }
    }
}
```