```
// Circular Linked List

#include <stdio.h>

#include <stdlib.h>

struct node
{
   int data;
   struct node * next;

}*head;

void createList(int n);

void displayList();

void insertAtBeginning(int data);

void insertAtN(int data, int position);

void deleteAll(struct node ** head, int key);

int  search(struct node *head, int key);

void update_element(int data);


void createList(int n)
{
   int i, data;
   struct node *prevNode, *newNode;

   if(n >= 1)
   {

      head = (struct node *)malloc(sizeof(struct node));

      printf("Enter data of 1 node: ");
      scanf("%d", &data);

      head->data = data;
      head->next = NULL;

      prevNode = head;


      for(i=2; i<=n; i++)
      {
         newNode = (struct node *)malloc(sizeof(struct node));

         printf("Enter data of %d node: ", i);
         scanf("%d", &data);
```

```
            newNode->data = data;
            newNode->next = NULL;


            prevNode->next = newNode;

            prevNode = newNode;
        }


        prevNode->next = head;

        printf("\nCIRCULAR LINKED LIST CREATED SUCCESSFULLY\n");
    }
}



void displayList()
{
    struct node *temp;
    int n = 1;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else
    {
        temp = head;
        printf("DATA IN THE LIST:\n");

        do
        {

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

            printf("->address %d , ",temp);

            printf("Next address= %d \n",temp->next);

            temp = temp->next;
            n++;
        }
        while(temp != head);
    }
}


void insertAtBeginning(int data)
{
    struct node *newNode, *current;

    if(head == NULL)
```

```
    {
        printf("List is empty.\n");
    }
    else
    {

        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = data;
        newNode->next = head;


        current = head;
        while(current->next != head)
        {
            current = current->next;
        }
        current->next = newNode;


        head = newNode;

        printf("NODE INSERTED SUCCESSFULLY\n");
    }
}



void insertAtN(int data, int position)
{
    struct node *newNode, *current;
    int i;

    if(head == NULL)
    {
        printf("List is empty.\n");
    }
    else if(position == 1)
    {
        insertAtBeginning(data);
    }
    else
    {

        newNode = (struct node *)malloc(sizeof(struct node));
        newNode->data = data;


        current = head;
        for(i=2; i<=position-1; i++)
        {
            current = current->next;
        }
```

```
        newNode->next = current->next;
        current->next = newNode;

        printf("NODE INSERTED SUCCESSFULLY.\n");
    }
}



void deleteAll(struct node ** head, int key)
{
    int i, count;
    struct node *prev, *cur;

    if (*head == NULL)
    {
        printf("List is empty.\n");
        return;
    }

    count = 0;
    cur   = *head;
    prev  = cur;


    while (prev->next != *head)
    {
        prev = prev->next;
        count++;
    }


    i = 0;
    while (i <= count)
    {
        if (cur->data == key)
        {

            if (cur->next != cur)
                prev->next = cur->next;
            else
                prev->next = NULL;


            if (cur == *head)
                *head = prev->next;


            free(cur);


            if (prev != NULL)
                cur = prev->next;
```

```
            else
                cur = NULL;
        }
        else
        {
            prev = cur;
            cur  = cur->next;
        }


        i++;

    }
}



int search(struct node *head, int key)
{
    int index = 0;
    struct node *current = head;

    // Iterate till end of list
    do
    {
        // Nothing to look into
        if (current == NULL)
            return;

        if (current->data == key)
            return index;

        current = current->next;
        index++;
    }
    while (current != head);


    return -1;
}
void update_element(int data)
{
    int count = 0;
    int update_ele;
    struct node* temp;
    temp = head;
    while(temp != NULL)
    {
        if(temp -> data == data)
        {
            printf("\nEnter the new data to update the old data : ");
            scanf("%d",&update_ele);

            temp -> data = update_ele;
```

```
            printf("\nHere\n");
            displayList();

        }
        else
        {
            count = count + 1;
            temp = temp -> next;
        }
    }
}




int main()
{
    int n, data, key, choice, index,  element;

    head = NULL;
     printf("CIRCULAR LINKED LIST   >\n");

        printf("1. To Create List.\n");
        printf("2. Display list\n");
        printf("3. Insert at beginning\n");
        printf("4. Insert at any position\n");
        printf("5. Delete all by key\n");
        printf("6. Search element\n");
        printf("7. updated in the list \n ");
        printf("0. Exit\n");

        printf("Enter your choice : ");

        scanf("%d", &choice);

    while(choice != 0)
    {

        switch(choice)
        {
        case 1:
            printf("Enter the total number of nodes in list: ");
            scanf("%d", &n);
            createList(n);
            break;
        case 2:
            displayList();
            break;
        case 3:
            printf("Enter data to be inserted at beginning: ");
            scanf("%d", &data);
            insertAtBeginning(data);
            break;
```

```
        case 4:
            printf("Enter node position: ");
            scanf("%d", &n);
            printf("Enter data you want to insert at %d position: ", n);
            scanf("%d", &data);
            insertAtN(data, n);
            break;

        case 5:
            printf("Enter key to delete from list: ");
            scanf("%d", &key);
            deleteAll(&head, key);
            break;
        case 6:
            printf("Enter element to search: ");
            scanf("%d", &n);
            index = search(head, n);

            if (index == -1)
                printf("%d not found in list.\n", n);
            else
                printf("%d found at %d position.\n", n, (index + 1));

            getchar(); // Hold screen
            getchar(); // Hold screen
            break;
        case 7:
            printf("\nEnter the element to be updated in the list : "); // value of the element to be searched in the list
            scanf("%d",&element);
            update_element(element);

            break;

        case 0:
            break;
        default:
            printf("Error! Invalid choice. Please choose between 0-7");
        }

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

    return 0;
}
```