Folder SEM 3\Exp4

1 printable files

(file list disabled)

SEM 3\Exp4\CDLL.c

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

// Node structure for the circular doubly linked list
struct Node
{
    int data;
    struct Node *next;
    struct Node *prev;
};

// Insert a node at the end of the circular doubly linked list
void insert(struct Node **head_ref, int new_data)
{
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;

    if (*head_ref == NULL)
    {
        new_node->next = new_node;
        new_node->prev = new_node;
        *head_ref = new_node;
        return;
    }

    struct Node *last = (*head_ref)->prev;

    new_node->next = *head_ref;
    (*head_ref)->prev = new_node;
    new_node->prev = last;
    last->next = new_node;
}

// Display the circular doubly linked list
void display(struct Node *head)
{
    if (head == NULL)
    {
        printf("List is empty.\n");
        return;
    }

    struct Node *temp = head;
    printf("Traversal in forward direction:\n");
    do
    {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("\n");

    printf("Traversal in reverse direction:\n");
    temp = head->prev;
    do
    {
        printf("%d ", temp->data);
        temp = temp->prev;
    } while (temp->next != head);
    printf("\n");
}

// Delete a node from the circular doubly linked list
void deleteNode(struct Node **head_ref, int key)
{
    if (*head_ref == NULL)
        return;

    struct Node *current = *head_ref;

    while (current->data != key)
    {
        current = current->next;
        if (current == *head_ref)
        {
            printf("Element %d not found in the list.\n", key);
            return;
        }
    }

    if (current->next == *head_ref && current->prev == *head_ref)
    {
        *head_ref = NULL;
        free(current);
        return;
    }

    if (current == *head_ref)
    {
        struct Node *last = (*head_ref)->prev;
        *head_ref = current->next;
        last->next = *head_ref;
        (*head_ref)->prev = last;
        free(current);
        return;
    }

    current->prev->next = current->next;
    current->next->prev = current->prev;

    free(current);
}

// Search for a specific value in the circular doubly linked list
void search(struct Node *head, int key)
{
    if (head == NULL)
    {
        printf("List is empty.\n");
        return;
    }

    struct Node *temp = head;
    int pos = 0;

    do
    {
        if (temp->data == key)
        {
            printf("Element %d found at position %d\n", key, pos);
            return;
        }
        temp = temp->next;
        pos++;
    } while (temp != head);

    printf("Element %d not found in the list\n", key);
}

// Count the number of nodes in the circular doubly linked list
int count(struct Node *head)
{
    if (head == NULL)
        return 0;

    struct Node *temp = head;
    int count = 0;

    do
    {
        count++;
        temp = temp->next;
    } while (temp != head);

    return count;
}

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

    while (1)
    {
        printf("\nCircular Doubly Linked List Operations:\n");
        printf("1. Insert\n");
        printf("2. Display\n");
        printf("3. Delete\n");
        printf("4. Search\n");
        printf("5. Count\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            insert(&head, value);
            break;
        case 2:
            display(head);
            break;
        case 3:
            printf("Enter the value to delete: ");
            scanf("%d", &key);
            deleteNode(&head, key);
            break;
        case 4:
            printf("Enter the value to search: ");
            scanf("%d", &key);
            search(head, key);
            break;
        case 5:
            printf("The number of nodes in the list: %d\n", count(head));
            break;
        case 6:
            exit(0);
        default:
            printf("Invalid choice!\n");
        }
    }

    return 0;
}
```