#include <stdio.h>

#include <stdlib.h>

typedef struct node {

int data;

struct node\* link;

} NODE;

typedef struct {

int count;

NODE\* head;

} LIST;

NODE\* createNode(int val) {

NODE\* n = (NODE\*)malloc(sizeof(NODE));

if (n == NULL) {

printf("Memory allocation failed\n");

return NULL;

}

n->data = val;

n->link = NULL;

return n;

}

NODE\* insert\_end(LIST\* list, int val) {

NODE\* n = createNode(val);

if (list->head == NULL) {

list->head = n;

n->link = n; // Point to itself in a circular list

} else {

NODE\* temp = list->head;

while (temp->link != list->head) {

temp = temp->link;

}

temp->link = n;

n->link = list->head;

}

list->count++;

return n;

}

NODE\* ins\_front(LIST\* list, int val) {

NODE\* n = createNode(val);

if (list->head == NULL) {

list->head = n;

n->link = n; // Point to itself in a circular list

} else {

NODE\* temp = list->head;

while (temp->link != list->head) {

temp = temp->link;

}

n->link = list->head;

list->head = n;

temp->link = n;

}

list->count++;

return n;

}

NODE\* insert\_position(LIST\* list, int position, int val) {

if (position < 1 || (position > list->count + 1)) {

printf("\nInvalid position\n");

return NULL;

}

NODE\* n = createNode(val);

list->count++;

if (position == 1) {

if (list->head == NULL) {

list->head = n;

n->link = n; // Point to itself in a circular list

} else {

NODE\* temp = list->head;

while (temp->link != list->head) {

temp = temp->link;

}

temp->link = n;

n->link = list->head;

list->head = n;

}

} else {

NODE\* temp = list->head;

for (int i = 1; i < position - 1; i++) {

temp = temp->link;

}

n->link = temp->link;

temp->link = n;

}

return n;

}

NODE\* delete\_front(LIST\* list) {

if (list->head == NULL) {

printf("\nThe list is empty\n");

return NULL;

}

NODE\* temp = list->head;

if (list->count == 1) {

printf("\n deleted element=%d\n", temp->data);

free(temp);

list->head = NULL;

} else {

NODE\* last = list->head;

while (last->link != list->head)

last = last->link;

list->head = list->head->link;

last->link = list->head;

printf("\n deleted element=%d\n", temp->data);

free(temp);

}

list->count--;

return list->head;

}

NODE\* delete\_rear(LIST\* list) {

if (list->head == NULL) {

printf("\nThe list is empty\n");

return NULL;

}

NODE\* current = list->head;

NODE\* previous = NULL;

while (current->link != list->head) {

previous = current;

current = current->link;

}

if (previous != NULL) {

previous->link = list->head; // Update the second-to-last node to point to the head

} else {

list->head = NULL; // If only one node is present

}

printf("\n deleted element=%d\n", current->data);

free(current);

list->count--;

return list->head;

}

NODE\* delete\_position(LIST\* list, int position) {

if (list->head == NULL || position < 1 || position > list->count) {

printf("\nInvalid position or empty list\n");

return NULL;

}

NODE\* current = list->head;

NODE\* previous = NULL;

if (position == 1) {

NODE\* last = list->head;

while (last->link != list->head) {

last = last->link;

}

if (last == list->head) {

printf("\n deleted element=%d\n", current->data);

free(current);

list->head = NULL;

return list->head;

}

list->head = current->link;

last->link = list->head;

printf("\n deleted element=%d\n", current->data);

free(current);

} else {

for (int i = 1; i < position; i++) {

previous = current;

current = current->link;

}

previous->link = current->link;

printf("\n deleted element=%d\n", current->data);

free(current);

}

return list->head;

}

NODE\* deleteByKey(LIST\* list, int key)

{

if (list->head == NULL)

{

printf("\nThe list is empty\n");

return NULL;

}

NODE\* current = list->head;

NODE\* previous = NULL;

do {

if (current->data == key)

{

// Node with the key found, delete it

if (previous == NULL)

{

// If the node to be deleted is the head

if (current->link == list->head)

{

// If it's the only node in the list

free(current);

list->head = NULL;

list->count = 0;

return NULL;

}

else

{

// Update head and link last node to the new head

NODE\* last = list->head;

while (last->link != list->head) {

last = last->link;

}

list->head = current->link;

last->link = list->head;

}

}

else

{

previous->link = current->link;

}

free(current);

list->count--;

printf("\nNode with key %d deleted\n", key);

return list->head;

}

// Move to the next node

previous = current;

current = current->link;

} while (current != list->head);

// If the key is not found in the list

printf("\nNode with key %d not found\n", key);

return list->head;

}

NODE\* searchByKey(LIST\* list, int key)

{

if (list->head == NULL)

{

// The list is empty, so there is nothing to search

printf("\nThe list is empty\n");

return NULL;

}

NODE\* current = list->head;

do {

if (current->data == key)

{

// Node with the key found, return the pointer to the node

printf("\nNode with key %d found\n", key);

return current;

}

// Move to the next node

current = current->link;

} while (current != list->head);

// If the key is not found in the list

printf("\nNode with key %d not found\n", key);

return NULL;

}

void reverse\_list(LIST\* list) {

if (list->head == NULL || list->head->link == list->head)

{

// Empty list or only one node, no change needed

return;

}

NODE\* prev = NULL;

NODE\* current = list->head;

NODE\* next = current->link;

do {

current->link = prev;

prev = current;

current = next;

next = current->link;

} while (current != list->head);

// Update the head to the last node, which becomes the new head after reversal

list->head->link = prev;

list->head = prev;

printf("\nCircular linked list reversed\n");

display(list);

}

LIST\* copyList(LIST\* list) {

if (list->head == NULL) {

printf("\nThe original list is empty\n");

return NULL;

}

LIST\* newList = (LIST\*)malloc(sizeof(LIST));

newList->count = 0;

newList->head = NULL;

NODE\* current = list->head;

do {

insert\_end(newList, current->data);

current = current->link;

} while (current != list->head);

return newList;

}

void create\_ordered(LIST \*list, int value)

{

NODE \*n = createNode(value);

NODE \*cur = list->head;

if (list->head == NULL ||value <= list->head->data)

{

list->head=ins\_front(list,value);

}

else

{

NODE \*temp = list->head;

NODE \*prev = NULL;

do

{

prev = temp;

temp = temp->link;

} while (temp != list->head && temp->data < value);

n->link = temp;

prev->link = n;

list->count++;

}

printf("\nOrdered Circular Linked List: \n");

display(list);

}

void display(LIST\* list) {

if (list->head == NULL) {

printf("\nThe list is empty\n");

return;

}

NODE\* current = list->head;

printf("List elements: ");

do {

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

current = current->link;

} while (current != list->head);

}

int main() {

int ch, val, position,key;

LIST\* list = (LIST\*)malloc(sizeof(LIST));

list->count = 0;

list->head = NULL;

LIST \*p= (LIST \*)malloc(sizeof(LIST));

p->head=NULL;

p->count=0;

NODE\* n;

for (;;) {

printf("\n1. Insert front\n2. insert\_end\n3. Ins\_pos\n4. Delete\_front\n5. Delete Rear\n6. Del pos\n7. Display\n8. CLL Reverse\n9. Copy CLL\n10. Create\_ordered CLL\n11.Search by key\n12.Delete by key\n13.Exit\n");

printf("Enter your choice: ");

scanf("%d", &ch);

switch (ch) {

case 1:

printf("Enter value to insert at front: ");

scanf("%d", &val);

list->head = ins\_front(list, val);

break;

case 2:

printf("Enter value to insert at end: ");

scanf("%d", &val);

n = insert\_end(list, val);

break;

case 3:

printf("\nEnter the position to insert: ");

scanf("%d", &position);

printf("Enter value to insert at position: ");

scanf("%d", &val);

n = insert\_position(list, position, val);

break;

case 4:

list->head = delete\_front(list);

break;

case 5:

delete\_rear(list);

break;

case 6:

printf("\nEnter the position to delete: ");

scanf("%d", &position);

n = delete\_position(list, position);

break;

case 7:

display(list);

break;

case 8:

reverse\_list(list);

//display(list);

break;

case 9:

LIST\* copiedList = copyList(list);

printf("\n Copied List\n");

display(copiedList);

break;

case 10:

printf("Creating ordered list...\n");

for (;;) {

printf("\nEnter -1 to stop or Enter value to continue: ");

scanf("%d", &val);

if (val == -1)

{

break;

}

create\_ordered(p, val);

}

break;

case 11:

printf("\nEnter the key to search: ");

scanf("%d", &key);

NODE\* result = searchByKey(list, key);

if (result != NULL)

{

// Node with the key found, you can access its data or perform other operations

printf("Node data: %d\n", result->data);

}

break;

case 12:

printf("Enter key to delete: ");

scanf("%d", &key);

list->head = deleteByKey(list, key);

break;

case 13:

printf("\nProgram exited\n");

free(list);

return 0;

default:

printf("\nInvalid Choice\n");

}

}

return 0;

}