#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

typedef struct {

int\* data;

int front,rear,size;

} CircularQueue;

// Function to initialize a circular queue

CircularQueue\* initializeQueue() {

CircularQueue\* queue = (CircularQueue\*)malloc(sizeof(CircularQueue));

if (queue == NULL) {

printf("Memory allocation failed.\n");

exit(EXIT\_FAILURE);

}

queue->data = (int\*)malloc(MAX\_SIZE \* sizeof(int));

if (queue->data == NULL) {

printf("Memory allocation failed.\n");

free(queue);

exit(EXIT\_FAILURE);

}

queue->front = -1;

queue->rear = -1;

queue->size = MAX\_SIZE;

return queue;

}

// Function to check if the queue is empty

int isQueueEmpty(CircularQueue\* queue) {

return queue->front == -1;

}

// Function to check if the queue is full

int isQueueFull(CircularQueue\* queue) {

return (queue->rear + 1) % queue->size == queue->front;

}

// Function to write data to the queue

void writeQueue(CircularQueue\* queue, int value) {

if (isQueueFull(queue)) {

// Queue is full, overwrite the oldest data

queue->front = (queue->front + 1) % queue->size;

}

// Move rear to the next position and write the value

queue->rear = (queue->rear + 1) % queue->size;

queue->data[queue->rear] = value;

// If the queue was empty, update front as well

if (isQueueEmpty(queue)) {

queue->front = queue->rear;

}

}

// Function to read data from the queue

int readQueue(CircularQueue\* queue) {

if (isQueueEmpty(queue)) {

printf("Queue is empty.\n");

return -1; // Returning a sentinel value to indicate an empty queue

}

int value = queue->data[queue->front];

// If there's only one element in the queue, reset front and rear

if (queue->front == queue->rear) {

queue->front = -1;

queue->rear = -1;

} else {

// Move front to the next position

queue->front = (queue->front + 1) % queue->size;

}

return value;

}

// Function to clear the queue

void clearQueue(CircularQueue\* queue) {

queue->front = -1;

queue->rear = -1;

}

// Function to display the contents of the queue

void displayQueue(CircularQueue\* queue) {

if (isQueueEmpty(queue)) {

printf("Queue is empty.\n");

return;

}

int i = queue->front;

do {

printf("%d ", queue->data[i]);

i = (i + 1) % queue->size;

} while (i != (queue->rear + 1) % queue->size);

printf("\n");

}

// Function to deallocate memory used by the queue

void freeQueue(CircularQueue\* queue) {

free(queue->data);

free(queue);

}

int main() {

CircularQueue\* queue = initializeQueue();

writeQueue(queue, 1);

writeQueue(queue, 2);

writeQueue(queue, 3);

displayQueue(queue);

int value = readQueue(queue);

printf("Read from queue: %d\n", value);

displayQueue(queue);

writeQueue(queue, 4);

writeQueue(queue, 5);

writeQueue(queue, 6);

displayQueue(queue);

freeQueue(queue);

return 0;

}