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

#define MAX_SIZE 5

typedef struct {
    int front, rear;
    int size;
    int *array;
} CircularQueue;

// Function to initialize a circular queue
void initialize_queue(CircularQueue *queue) {
    queue->front = queue->rear = -1;
    queue->size = MAX_SIZE;
    queue->array = (int *)malloc(MAX_SIZE * sizeof(int));
}

// Function to check if the queue is empty
int is_empty(CircularQueue *queue) {
    return (queue->front == -1);
}

// Function to check if the queue is full
int is_full(CircularQueue *queue) {
    return ((queue->rear + 1) % queue->size == queue->front);
}

// Function to write data to the queue
void write_queue(CircularQueue *queue, int data) {
    if (is_empty(queue)) {
        queue->front = 0;
        queue->rear = 0;
        queue->array[queue->rear] = data;
    } else if (is_full(queue)) {
        // Overwrite the oldest data
        queue->front = (queue->front + 1) % queue->size;
        queue->rear = (queue->rear + 1) % queue->size;
        queue->array[queue->rear] = data;
    } else {
        queue->rear = (queue->rear + 1) % queue->size;
        queue->array[queue->rear] = data;
    }
}
```

```
// Function to read data from the queue
int read_queue(CircularQueue *queue) {
    int data = -1; // Default value if the queue is empty

    if (!is_empty(queue)) {
        data = queue->array[queue->front];

        if (queue->front == queue->rear) {
            // Reset the queue when the last element is read
            queue->front = queue->rear = -1;
        } else {
            queue->front = (queue->front + 1) % queue->size;
        }
    }

    return data;
}

// Function to clear the queue
void clear_queue(CircularQueue *queue) {
    queue->front = queue->rear = -1;
}

// Function to print the elements of the queue
void print_queue(CircularQueue *queue) {
    if (is_empty(queue)) {
        printf("Queue is empty.\n");
        return;
    }

    int i = queue->front;
    printf("Queue elements: ");
    do {
        printf("%d ", queue->array[i]);
        i = (i + 1) % queue->size;
    } while (i != (queue->rear + 1) % queue->size);
    printf("\n");
}

// Function to free the memory allocated for the queue
void destroy_queue(CircularQueue *queue) {
    free(queue->array);
}
```

```
int main() {
    CircularQueue queue;
    initialize_queue(&queue);

    write_queue(&queue, 1);
    write_queue(&queue, 2);
    write_queue(&queue, 3);
    write_queue(&queue, 4);
    print_queue(&queue);

    // Queue is full, so the next write will overwrite the oldest data
    write_queue(&queue, 5);
    print_queue(&queue);

    // Read from the queue
    int data = read_queue(&queue);
    printf("Read from the queue: %d\n", data);
    print_queue(&queue);

    // Clear the queue
    clear_queue(&queue);
    print_queue(&queue);

    // Cleanup
    destroy_queue(&queue);

    return 0;
}
```