QUESTION1

include <stdio.h>

#define MAX\_SIZE 100

struct Queue {

int items[MAX\_SIZE];

int front;

int rear;

};

void initQueue(struct Queue \*q) {

q->front = -1;

q->rear = -1;

}

int isEmpty(struct Queue \*q) {

if (q->rear == -1)

return 1;

else

return 0;

}

int isFull(struct Queue \*q) {

if (q->rear == MAX\_SIZE - 1)

return 1;

else

return 0;

}

void enqueue(struct Queue \*q, int value) {

if (isFull(q)) {

printf("Queue is full\n");

} else {

if (q->front == -1) q->front = 0;

q->rear++;

q->items[q->rear] = value;

}

}

int dequeue(struct Queue \*q) {

int item;

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

} else {

item = q->items[q->front];

q->front++;

if (q->front > q->rear) {

q->front = q->rear = -1;

}

return item;

}

}

int find(struct Queue \*q, int value) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return -1;

} else {

for (int i = q->front; i <= q->rear; i++) {

if (q->items[i] == value) {

return i;

}

}

printf("%d not found in the queue\n", value);

return -1;

}

}

int main() {

struct Queue q;

initQueue(&q);

enqueue(&q, 67);

enqueue(&q, 2);

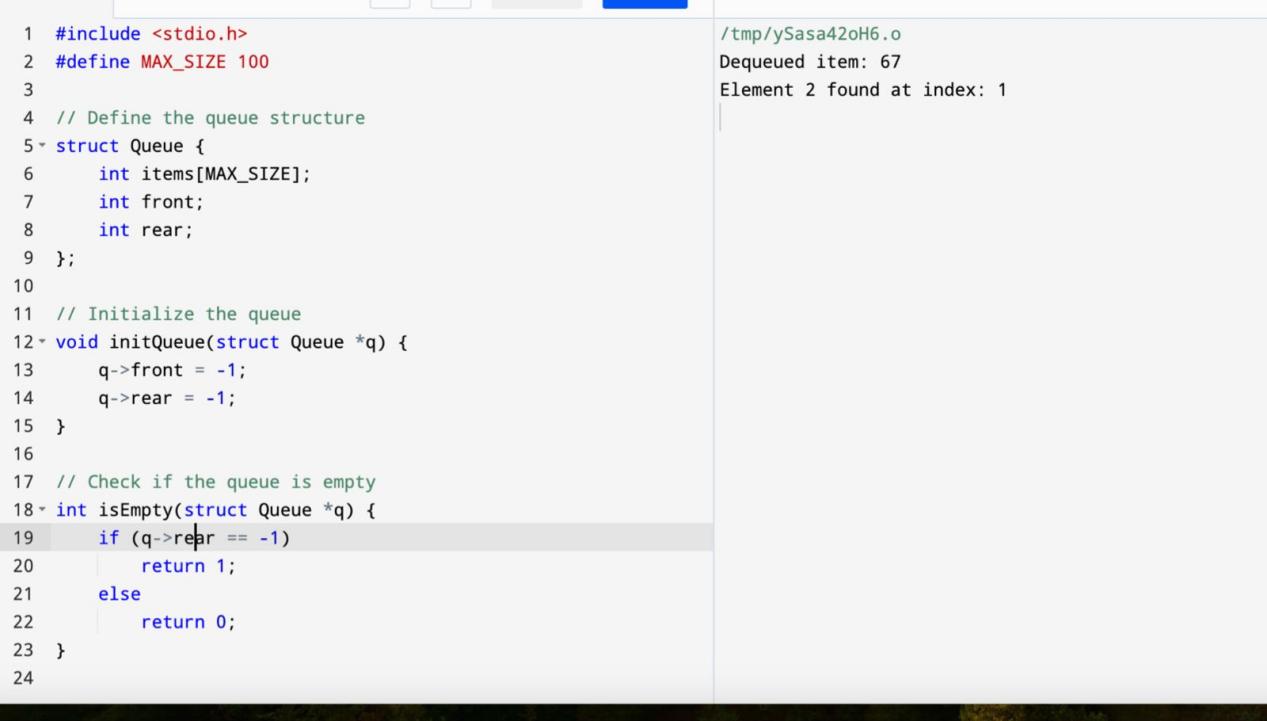
enqueue(&q, 78);

printf("Dequeued item: %d\n", dequeue(&q));

printf("Element 2 found at index: %d\n", find(&q, 2));

return 0;

}



QUESTION2

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 10

typedef struct {

int items[MAX\_SIZE];

int front, rear;

} CircularQueue;

void initializeQueue(CircularQueue \*q) {

q->front = -1;

q->rear = -1;

}

int isEmpty(CircularQueue \*q) {

return (q->front == -1 && q->rear == -1);

}

int isFull(CircularQueue \*q) {

return ((q->rear + 1) % MAX\_SIZE == q->front);

}

void enqueue(CircularQueue \*q, int data) {

if (isFull(q)) {

printf("Queue is full\n");

return;

}

if (isEmpty(q)) {

q->front = 0;

q->rear = 0;

} else {

q->rear = (q->rear + 1) % MAX\_SIZE;

}

q->items[q->rear] = data;

}

void dequeue(CircularQueue \*q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

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

q->front = -1;

q->rear = -1;

} else {

q->front = (q->front + 1) % MAX\_SIZE;

}

}

int peek(CircularQueue \*q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

exit(EXIT\_FAILURE);

}

return q->items[q->front];

}

void display(CircularQueue \*q) {

if (isEmpty(q)) {

printf("Queue is empty\n");

return;

}

int i = q->front;

printf("Queue elements: ");

do {

printf("%d ", q->items[i]);

i = (i + 1) % MAX\_SIZE;

} while (i != (q->rear + 1) % MAX\_SIZE);

printf("\n");

}

int main() {

CircularQueue q;

initializeQueue(&q);

enqueue(&q, 1);

enqueue(&q, 2);

enqueue(&q, 3);

enqueue(&q, 4);

enqueue(&q, 5);

display(&q);

dequeue(&q);

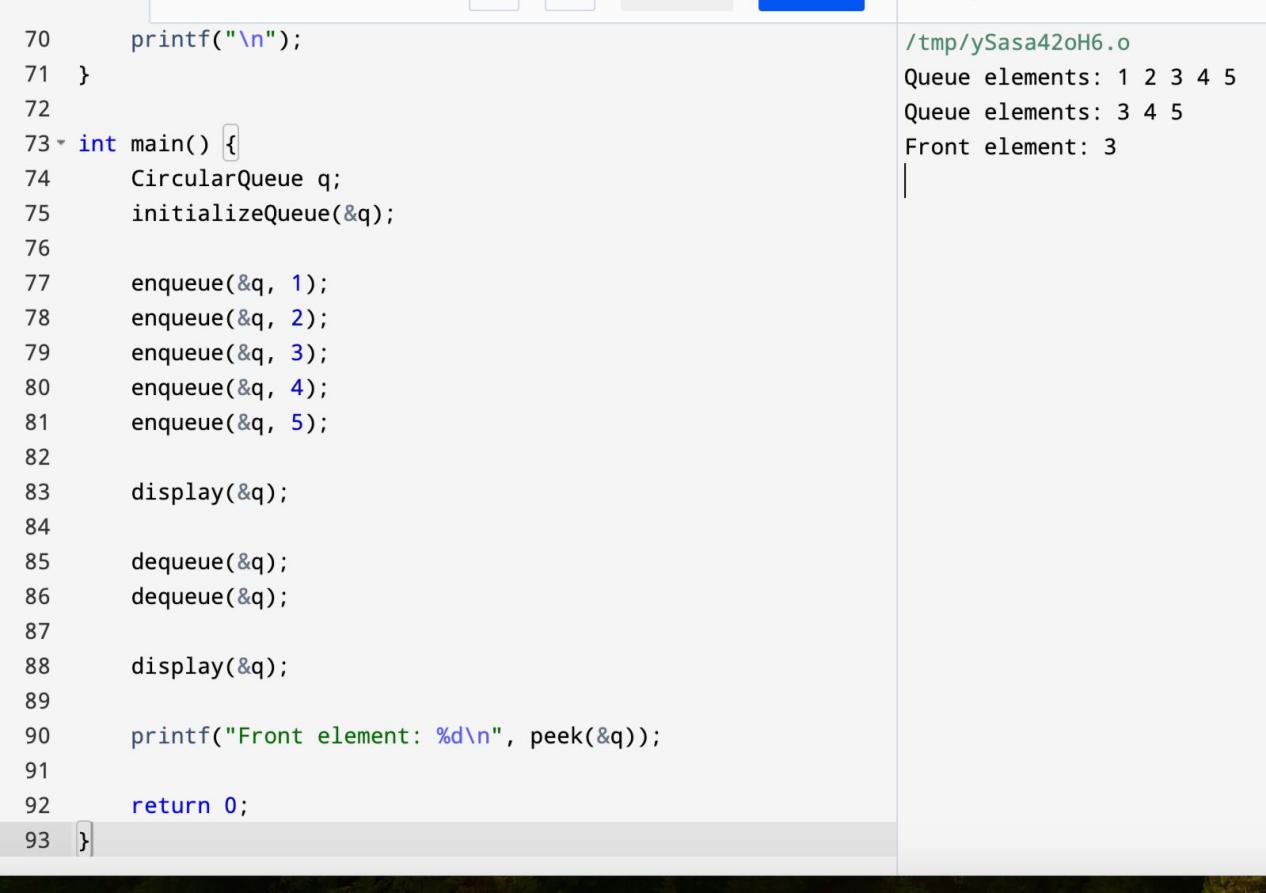
dequeue(&q);

display(&q);

printf("Front element: %d\n", peek(&q));

return 0;

}



QUESTION3

#include <stdio.h>

#include <stdbool.h>

#define N 4

bool isSafe(int board[N][N], int row, int col) {

int i, j;

for (i = 0; i < col; i++)

if (board[row][i])

return false;

for (i = row, j = col; i >= 0 && j >= 0; i--, j--)

if (board[i][j])

return false;

for (i = row, j = col; j >= 0 && i < N; i++, j--)

if (board[i][j])

return false;

return true;

}

bool solveNQUtil(int board[N][N], int col) {

if (col >= N)

return true;

for (int i = 0; i < N; i++) {

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0;

}

}

return false;

}

bool solveNQ() {

int board[N][N] = {{0, 0, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}};

if (solveNQUtil(board, 0) == false) {

printf("Solution does not exist");

return false;

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++)

printf(" %d ", board[i][j]);

printf("\n");

}

return true;

}

int main() {

solveNQ();

return 0;

}

