1.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

struct Queue {

int items[MAX\_SIZE];

int front;

int rear;

};

struct Queue\* createQueue() {

struct Queue\* queue = (struct Queue\*)malloc(sizeof(struct Queue));

queue->front = -1; // Initialize front

queue->rear = -1; // Initialize rear

return queue;

}

int isFull(struct Queue\* queue) {

return (queue->rear == MAX\_SIZE - 1);

}

int isEmpty(struct Queue\* queue) {

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

}

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

if (isFull(queue)) {

printf("Queue is full!\n");

return;

}

if (isEmpty(queue)) {

queue->front = queue->rear = 0;

} else {

queue->rear++;

}

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

printf("%d enqueued to queue.\n", value);

}

int dequeue(struct Queue\* queue) {

int removedItem;

if (isEmpty(queue)) {

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

return -1;

}

removedItem = queue->items[queue->front];

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

queue->front = queue->rear = -1;

} else {

queue->front++;

}

printf("%d dequeued from queue.\n", removedItem);

return removedItem;

}

int findElement(struct Queue\* queue, int value) {

if (isEmpty(queue)) {

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

return -1;

}

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

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

printf("%d found at position %d.\n", value, i - queue->front + 1);

return i - queue->front + 1;

}

}

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

return -1;

}

int main() {

struct Queue\* queue = createQueue();

enqueue(queue, 10);

enqueue(queue, 20);

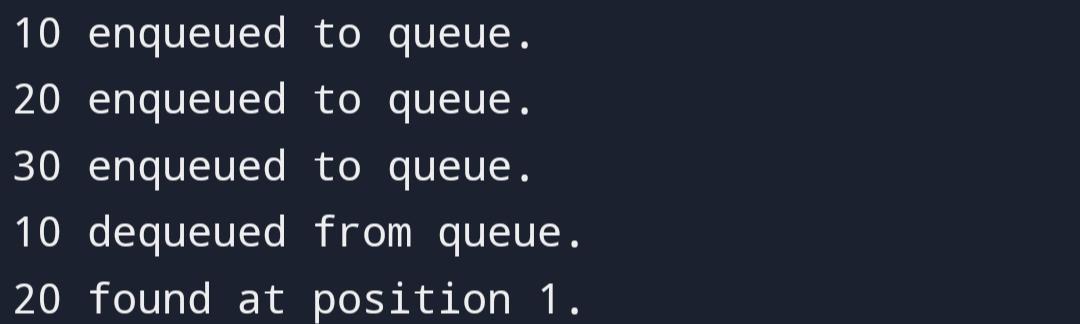
enqueue(queue, 30);

dequeue(queue);

findElement(queue, 20);

return 0;

}



2.

#include <stdio.h>

#define MAX\_SIZE 10

int queue[MAX\_SIZE];

int front = -1, rear = -1;

int isEmpty() {

return front == -1 && rear == -1;

}

int isFull() {

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

}

void enqueue(int data) {

if (isFull()) {

printf("Queue is full. Cannot enqueue.\n");

return;

} else if (isEmpty()) {

front = rear = 0;

} else {

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

}

queue[rear] = data;

}

int dequeue() {

if (isEmpty()) {

printf("Queue is empty. Cannot dequeue.\n");

return -1;

} else if (front == rear) {

int temp = queue[front];

front = rear = -1;

return temp;

} else {

int temp = queue[front];

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

return temp;

}

}

int findElement(int element) {

int i;

for (i = front; i != (rear + 1) % MAX\_SIZE; i = (i + 1) % MAX\_SIZE) {

if (queue[i] == element) {

return 1; // Element found

}

}

return 0;

}

int main() {

enqueue(1);

enqueue(2);

enqueue(3);

printf("Dequeued element: %d\n", dequeue());

enqueue(4);

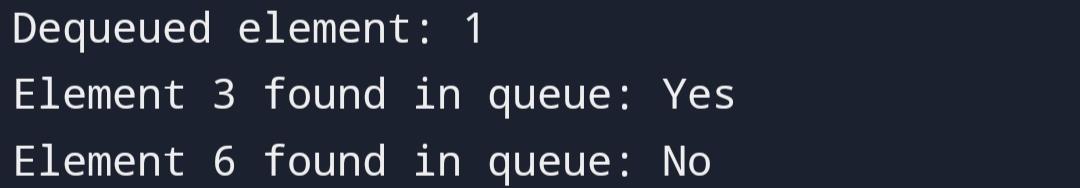
enqueue(5);

printf("Element 3 found in queue: %s\n", findElement(3) ? "Yes" : "No");

printf("Element 6 found in queue: %s\n", findElement(6) ? "Yes" : "No");

return 0;

}



3.

#include <stdio.h>

#define N 4

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

void printSolution() {

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

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

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

}

printf("\n");

}

}

int isSafe(int row, int col) {

int i, j;

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

if (board[row][i])

return 0;

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

if (board[i][j])

return 0;

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

if (board[i][j])

return 0;

return 1;

}

int solveNQUtil(int col) {

if (col >= N)

return 1;

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

if (isSafe(i, col)) {

board[i][col] = 1;

if (solveNQUtil(col + 1))

return 1;

board[i][col] = 0; // backtrack

}

}

return 0;

}

int solveNQ() {

if (solveNQUtil(0) == 0) {

printf("Solution does not exist");

return 0;

}

printSolution();

return 1;

}

int main() {

solveNQ();

return 0;

}

