Lab task 5

1. #include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 100

typedef struct {

int items[MAX\_SIZE];

int front;

int rear;

} Queue;

Queue\* createQueue() {

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

queue->front = -1;

queue->rear = -1;

return queue;

}

int isFull(Queue\* queue) {

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

}

int isEmpty(Queue\* queue) {

return queue->front == -1;

}

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

if (isFull(queue)) {

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

} else {

if (isEmpty(queue)) {

queue->front = 0;

}

queue->rear++;

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

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

}

}

int dequeue(Queue\* queue) {

int item;

if (isEmpty(queue)) {

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

return -1;

} else {

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

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

queue->front = -1;

queue->rear = -1;

} else {

queue->front++;

}

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

return item;

}

}

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

if (isEmpty(queue)) {

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

return -1;

} else {

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

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

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

return i - queue->front + 1;

}

}

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

return -1;

}

}

int main() {

Queue\* queue = createQueue();

enqueue(queue, 10);

enqueue(queue, 20);

enqueue(queue, 30);

dequeue(queue);

dequeue(queue);

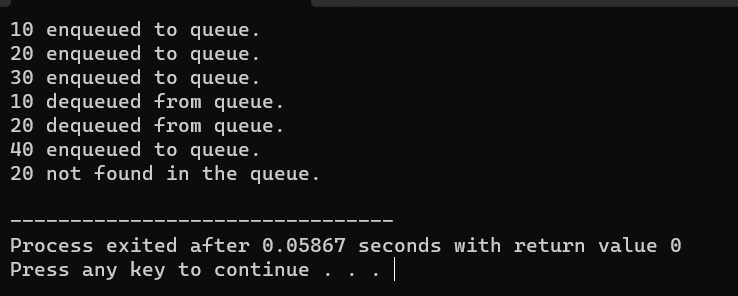
enqueue(queue, 40);

findElement(queue, 20);

return 0;

}

Output:



2. #include <stdio.h>

#define MAX\_SIZE 10

int front = -1, rear = -1;

int queue[MAX\_SIZE];

int isFull() {

if ((front == 0 && rear == MAX\_SIZE - 1) || (rear == front - 1))

return 1;

else

return 0;

}

int isEmpty() {

if (front == -1)

return 1;

else

return 0;

}

void enqueue(int data) {

if (isFull())

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

else {

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

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

queue[rear] = data;

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

}

}

int dequeue() {

int data;

if (isEmpty()) {

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

return -1;

} else {

data = queue[front];

if (front == rear) {

front = -1;

rear = -1;

} else {

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

}

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

return data;

}

}

int find(int element) {

if (isEmpty()) {

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

return -1;

} else {

int i;

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

if (queue[i] == element) {

printf("%d found at position %d.\n", element, i);

return i;

}

}

if (queue[i] == element) {

printf("%d found at position %d.\n", element, i);

return i;

}

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

return -1;

}

}

int main() {

enqueue(1);

enqueue(2);

enqueue(3);

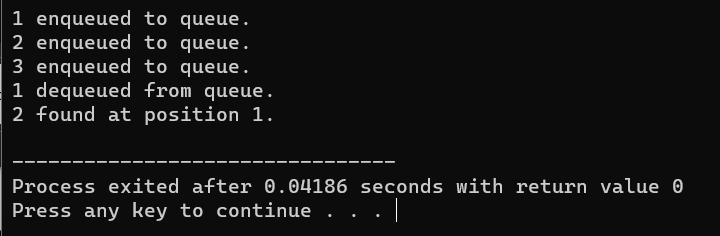
dequeue();

find(2);

return 0;

}

Output:



3. #include <stdio.h>

#include <stdbool.h>

#define N 4

void printSolution(int board[N][N]) {

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

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

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

printf("\n");

}

}

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;

}

printSolution(board);

return true;

}

int main() {

solveNQ();

return 0;

}

Output: