1. Week 3 &amp; 4: Simulate the following tasks:

a. Implementation the following operations: enqueue, dequeue and finding an element:

1. Linear Queue using arrays

2. Circular queue using arrays

1.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

typedef struct {

int queue[MAX\_SIZE];

int front, rear;

} LinearQueue;

LinearQueue\* createLinearQueue() {

LinearQueue\* q = (LinearQueue\*)malloc(sizeof(LinearQueue));

q->front = -1;

q->rear = -1;

return q;

}

int isLinearQueueEmpty(LinearQueue\* q) {

return q->front == -1;

}

int isLinearQueueFull(LinearQueue\* q) {

return (q->rear == MAX\_SIZE - 1) ? 1 : 0;

}

void linearEnqueue(LinearQueue\* q, int data) {

if (isLinearQueueFull(q)) {

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

return;

}

if (q->front == -1) {

q->front = 0;

}

q->rear++;

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

}

int linearDequeue(LinearQueue\* q) {

if (isLinearQueueEmpty(q)) {

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

return -1;

}

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

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

q->front = -1;

q->rear = -1;

} else {

q->front++;

}

return data;

}

int main() {

LinearQueue\* q = createLinearQueue();

linearEnqueue(q, 1);

linearEnqueue(q, 2);

linearEnqueue(q, 3);

linearEnqueue(q, 4);

linearEnqueue(q, 5);

printf("Dequeue: %d\n", linearDequeue(q));

printf("Dequeue: %d\n", linearDequeue(q));

free(q);

return 0;

}



2.

#include <stdio.h>

#include <stdlib.h>

#define MAX\_SIZE 5

typedef struct {

int queue[MAX\_SIZE];

int front, rear;

} CircularQueue;

CircularQueue\* createCircularQueue() {

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

q->front = -1;

q->rear = -1;

return q;

}

int isCircularQueueEmpty(CircularQueue\* q) {

return q->front == -1;

}

int isCircularQueueFull(CircularQueue\* q) {

return (q->rear + 1) % MAX\_SIZE == q->front ? 1 : 0;

}

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

if (isCircularQueueFull(q)) {

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

return;

}

if (isCircularQueueEmpty(q)) {

q->front = 0;

}

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

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

}

int circularDequeue(CircularQueue\* q) {

if (isCircularQueueEmpty(q)) {

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

return -1;

}

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

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

q->front = -1;

q->rear = -1;

} else {

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

}

return data;

}

int main() {

CircularQueue\* q = createCircularQueue();

circularEnqueue(q, 1);

circularEnqueue(q, 2);

circularEnqueue(q, 3);

circularEnqueue(q, 4);

circularEnqueue(q, 5);

printf("Dequeue: %d\n", circularDequeue(q));

printf("Dequeue: %d\n", circularDequeue(q));

free(q);

return 0;

}

A screenshot of a computer

Description automatically generated

ASSIGNMENT-2

#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 solveNQueensUtil(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 (solveNQueensUtil(board, col + 1))

return true;

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

}

}

return false;

}

bool solveNQueens() {

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

{0, 0, 0, 0},

{0, 0, 0, 0},

{0, 0, 0, 0}};

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

printf("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

int main() {

solveNQueens();

return 0;

}

A screenshot of a computer output

Description automatically generated