LAB ASSIGNMENT 5

U24CS076

RUSHANG BAGADA

Q1: 1. Reversing a Queue using another Queue.

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

// Queue structure

struct Queue

{

    int items[MAX];

    int front, rear;

};

struct Queue \*createQueue()

{

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

    queue->front = -1;

    queue->rear = -1;

    return queue;

}

int isEmpty(struct Queue \*queue)

{

    return queue->front == -1;

}

int isFull(struct Queue \*queue)

{

    return queue->rear == MAX - 1;

}

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

{

    if (isFull(queue))

    {

        printf("Queue is full!\n");

        return;

    }

    if (isEmpty(queue))

    {

        queue->front = 0;

    }

    queue->rear++;

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

}

int dequeue(struct Queue \*queue)

{

    if (isEmpty(queue))

    {

        printf("Queue is empty!\n");

        return -1;

    }

    int dequeuedValue = queue->items[queue->front];

    if (queue->front == queue->rear)

    {

        queue->front = queue->rear = -1; // Reset the queue

    }

    else

    {

        queue->front++;

    }

    return dequeuedValue;

}

void reverseQueue(struct Queue \*originalQueue)

{

    struct Queue \*tempQueue = createQueue();

    while (!isEmpty(originalQueue))

    {

        int frontElement = dequeue(originalQueue);

        int size = tempQueue->rear - tempQueue->front + 1;

        enqueue(tempQueue, frontElement);

        for (int i = 0; i < size; i++)

        {

            enqueue(tempQueue, dequeue(tempQueue));

        }

    }

    while (!isEmpty(tempQueue))

    {

        enqueue(originalQueue, dequeue(tempQueue));

    }

    free(tempQueue);

}

void printQueue(struct Queue \*queue)

{

    if (isEmpty(queue))

    {

        printf("Queue is empty!\n");

        return;

    }

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

    {

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

    }

    printf("\n");

}

int main()

{

    struct Queue \*originalQueue = createQueue();

    enqueue(originalQueue, 1);

    enqueue(originalQueue, 2);

    enqueue(originalQueue, 3);

    enqueue(originalQueue, 4);

    enqueue(originalQueue, 5);

    printf("Original Queue: ");

    printQueue(originalQueue);

    reverseQueue(originalQueue);

    printf("Reversed Queue: ");

    printQueue(originalQueue);

    free(originalQueue);

    return 0;

}

OUTPUT:



2. Create a system to manage orders in a restaurant using a circular queue. The system

should allow customers to place orders, kitchen staff to process orders, and waitstaff to

deliver orders to tables, all while efficiently managing the queue of orders with a circular

structure.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_ORDERS 5

typedef struct

{

    int orderID;

    char customerName[50];

    char foodItem[50];

} Order;

typedef struct

{

    Order orders[MAX\_ORDERS];

    int front, rear, count;

} CircularQueue;

void initializeQueue(CircularQueue \*q)

{

    q->front = 0;

    q->rear = -1;

    q->count = 0;

}

int isFull(CircularQueue \*q)

{

    return (q->count == MAX\_ORDERS);

}

int isEmpty(CircularQueue \*q)

{

    return (q->count == 0);

}

void placeOrder(CircularQueue \*q)

{

    if (isFull(q))

    {

        printf("Order queue is full! Please wait for pending orders to be processed.\n");

        return;

    }

    Order newOrder;

    printf("Enter Order ID: ");

    scanf("%d", &newOrder.orderID);

    printf("Enter Customer Name: ");

    scanf(" *%*[^\n]", newOrder.customerName);

    printf("Enter Food Item: ");

    scanf(" *%*[^\n]", newOrder.foodItem);

    q->rear = (q->rear + 1) % MAX\_ORDERS; // Circular increment

    q->orders[q->rear] = newOrder;

    q->count++;

    printf("Order placed successfully!\n");

}

void processOrder(CircularQueue \*q)

{

    if (isEmpty(q))

    {

        printf("No pending orders!\n");

        return;

    }

    Order processedOrder = q->orders[q->front];

    printf("\nProcessing Order...\n");

    printf("Order ID: %d | Customer: %s | Food: %s\n",

           processedOrder.orderID, processedOrder.customerName, processedOrder.foodItem);

    q->front = (q->front + 1) % MAX\_ORDERS; // Circular increment

    q->count--;

    printf("Order Processed!\n");

}

void displayOrders(CircularQueue \*q)

{

    if (isEmpty(q))

    {

        printf("No orders in the queue.\n");

        return;

    }

    printf("\nCurrent Orders in Queue:\n");

    int index = q->front;

    for (int i = 0; i < q->count; i++)

    {

        printf("Order ID: %d | Customer: %s | Food: %s\n",

               q->orders[index].orderID, q->orders[index].customerName, q->orders[index].foodItem);

        index = (index + 1) % MAX\_ORDERS; // Circular increment

    }

}

int main()

{

    CircularQueue orderQueue;

    initializeQueue(&orderQueue);

    int choice;

    while (1)

    {

        printf("\n----- Restaurant Order Management -----\n");

        printf("1. Place Order\n");

        printf("2. Process Order\n");

        printf("3. View Pending Orders\n");

        printf("4. Exit\n");

        printf("Enter choice: ");

        scanf("%d", &choice);

        switch (choice)

        {

        case 1:

            placeOrder(&orderQueue);

            break;

        case 2:

            processOrder(&orderQueue);

            break;

        case 3:

            displayOrders(&orderQueue);

            break;

        case 4:

            printf("Exiting system. Have a great day!\n");

            return 0;

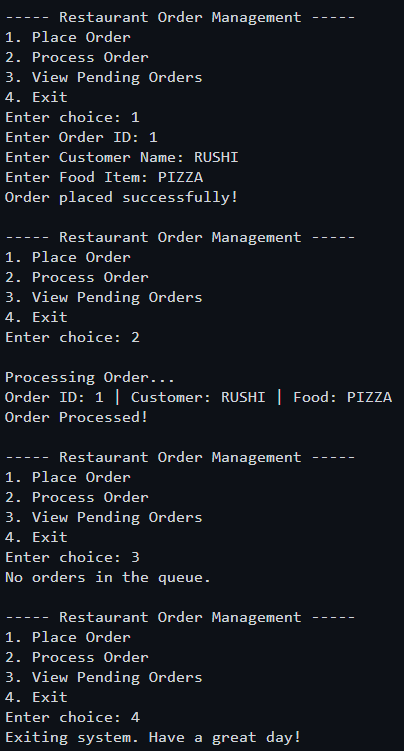
        default:

            printf("Invalid choice! Please try again.\n");

        }

    }

}



3. Implement a program that takes a string input and uses a DEQUE (doubly ended queue)

to check if the string is a palindrome. The program should utilize the properties of a deque

to efficiently compare characters from both ends of the string.

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

// Structure for Deque (Doubly Ended Queue)

typedef struct

{

    char items[MAX];

    int front, rear;

} Deque;

// Function to initialize the deque

void initializeDeque(Deque \*dq)

{

    dq->front = -1;

    dq->rear = -1;

}

// Function to check if the deque is empty

int isEmpty(Deque \*dq)

{

    return (dq->front == -1);

}

// Function to check if the deque is full

int isFull(Deque \*dq)

{

    return ((dq->rear + 1) % MAX == dq->front);

}

// Function to insert at the rear

void insertRear(Deque \*dq, char ch)

{

    if (isFull(dq))

    {

        printf("Deque is full!\n");

        return;

    }

    if (isEmpty(dq))

    {

        dq->front = dq->rear = 0;

    }

    else

    {

        dq->rear = (dq->rear + 1) % MAX;

    }

    dq->items[dq->rear] = ch;

}

// Function to delete from the front

char deleteFront(Deque \*dq)

{

    if (isEmpty(dq))

    {

        printf("Deque is empty!\n");

        return '\0';

    }

    char ch = dq->items[dq->front];

    if (dq->front == dq->rear)

    {

        dq->front = dq->rear = -1; // Reset when last element is removed

    }

    else

    {

        dq->front = (dq->front + 1) % MAX;

    }

    return ch;

}

// Function to delete from the rear

char deleteRear(Deque \*dq)

{

    if (isEmpty(dq))

    {

        printf("Deque is empty!\n");

        return '\0';

    }

    char ch = dq->items[dq->rear];

    if (dq->front == dq->rear)

    {

        dq->front = dq->rear = -1; // Reset when last element is removed

    }

    else

    {

        dq->rear = (dq->rear - 1 + MAX) % MAX;

    }

    return ch;

}

// Function to check if a string is a palindrome using a deque

int isPalindrome(char str[])

{

    Deque dq;

    initializeDeque(&dq);

    int len = strlen(str);

    // Insert characters into deque (ignoring case and non-alphabetic characters)

    for (int i = 0; i < len; i++)

    {

            insertRear(&dq, str[i]);

    }

    // Check palindrome by comparing front and rear characters

    while (dq.front < dq.rear)

    {

        if (deleteFront(&dq) != deleteRear(&dq))

        {

            return 0; // Not a palindrome

        }

    }

    return 1; // It's a palindrome

}

// Main function

int main()

{

    char str[MAX];

    printf("Enter a string: ");

    fgets(str, MAX, stdin);

    str[strcspn(str, "\n")] = '\0'; // Remove newline character from input

    if (isPalindrome(str))

    {

        printf("The string is a palindrome!\n");

    }

    else

    {

        printf("The string is NOT a palindrome.\n");

    }

    return 0;

}

