Student Information

- **Name**: [ATUL KUMAR]
- **Student ID**: [590011190]
- **Branch**: [MCA]
- **Batch**: [B1]
- **Instructor**: [Dr. Sourbh Kumar]
- **Date**: [28/10/2024]

Data Structure...

Lab Assignment 1: Queue Implementation Using Arrays

Problem Statement: Implement a queue data structure using arrays. Your program should support the following queue operations:

1. Enqueue: Add an element to the rear of the queue.
2. Dequeue: Remove an element from the front of the queue.
3. Peek: Display the front element without removing it.
4. IsEmpty: Check if the queue is empty.
5. IsFull: Check if the queue is full (assume a fixed size).

Assignment Tasks:

- Write a C program that defines a queue using arrays.
- Implement the queue operations mentioned above.
- Demonstrate queue overflow and underflow conditions.
- Write a main program to test all queue operations.

Solution:-

```
#include <stdio.h>
#define MAX 5 // Define a fixed size for the queue
// Queue structure definition
struct Queue {
int items[MAX];
int front, rear;
};
// Initialize the queue
void initializeQueue(struct Queue* queue) {
queue->front = -1;
queue->rear = -1;
}
// Check if the queue is full
int isFull(struct Queue* queue) {
return queue->rear == MAX - 1;
}
```

```
// Check if the queue is empty
int isEmpty(struct Queue* queue) {
return queue->front == -1 || queue->front > queue->rear;
}
// Enqueue an element into the queue
void enqueue(struct Queue* queue, int value) {
if (isFull(queue)) {
printf("Queue Overflow! Cannot enqueue %d\n", value);
} else {
if (queue->front == -1) queue->front = 0;
queue->items[++queue->rear] = value;
printf("Enqueued %d\n", value);
}
}
// Dequeue an element from the queue
int dequeue(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue Underflow! No elements to dequeue\n");
return -1;
} else {
return queue->items[queue->front++];
}
}
// Peek at the front element of the queue
int peek(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty! No front element\n");
return -1;
} else {
return queue->items[queue->front];
}
}
int main() {
struct Queue queue;
initializeQueue(&queue);
// Demonstrate queue operations
enqueue(&queue, 10);
enqueue(&queue, 20);
```

```
enqueue(&queue, 30);
enqueue(&queue, 40);
enqueue(&queue, 50);
// Attempt to enqueue when queue is full
enqueue(&queue, 60);
// Peek at the front element
printf("Front element: %d\n", peek(&queue));
// Dequeue elements from the queue
printf("Dequeued %d\n", dequeue(&queue));
printf("Dequeued %d\n", dequeue(&queue));
// Display the front element again
printf("Front element: %d\n", peek(&queue));
// Dequeue all elements
while (!isEmpty(&queue)) {
printf("Dequeued %d\n", dequeue(&queue));
}
// Attempt to dequeue from an empty queue
dequeue(&queue);
return 0;
}
```

Output:-

```
Enqueued 10
Enqueued 20
Enqueued 30
Enqueued 40
Enqueued 50
Queue Overflow! Cannot enqueue 60
Front element: 10
Dequeued 10
Dequeued 20
Front element: 30
Dequeued 30
Dequeued 40
Dequeued 50
Queue Underflow! No elements to dequeue

e:\MCA\MCA 24-25\DSA\practical>
```

Lab Assignment 2: Queue Implementation Using Linked Lists

Problem Statement:Implement a queue data structure using a linked list. Your program should support the following operations:

1. **Enqueue: Add an element to the rear of the queue.**
2. **Dequeue: Remove an element from the front of the queue.**
3. **Peek: Display the front element without removing it.**
4. **IsEmpty: Check if the queue is empty.**

Assignment Tasks:

- **Write a C program that defines a queue using a singly linked list.**
- **Implement the queue operations mentioned above.**
- **Demonstrate queue operations using linked lists.**
- **Write a main program to test all queue operations**

Soltuion:-

```
#include <stdio.h>
#include <stdlib.h>
// Define a node structure for the queue
struct Node {
int data;
struct Node* next;
};
// Define the queue structure
struct Queue {
struct Node* front;
struct Node* rear;
};
// Function to initialize the queue
void initializeQueue(struct Queue* queue) {
queue->front = NULL;
queue->rear = NULL;
}
// Check if the queue is empty
int isEmpty(struct Queue* queue) {
return queue->front == NULL;
}
// Enqueue an element into the queue
void enqueue(struct Queue* queue, int value) {
```

```
struct Node* newNode = (struct
Node*)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;
if (queue->rear == NULL) { // If queue is empty
queue->front = newNode;
queue->rear = newNode;
} else {
queue->rear->next = newNode; // Link the new node at
the end
queue->rear = newNode; // Update the rear to the new
node
}
printf("Enqueued %d\n", value);
}
// Dequeue an element from the queue
int dequeue(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue Underflow! No elements to dequeue.\n");
return -1;
}
struct Node* temp = queue->front;
int dequeuedValue = temp->data;
queue->front = queue->front->next;
if (queue->front == NULL) { // If the queue becomes empty
queue->rear = NULL;
}
free(temp);
return dequeuedValue;
}
// Peek at the front element of the queue
int peek(struct Queue* queue) {
if (isEmpty(queue)) {
printf("Queue is empty! No front element.\n");
return -1;
}
return queue->front->data;
}
int main() {
```

```
struct Queue queue;
initializeQueue(&queue);
// Demonstrate queue operations
enqueue(&queue, 10);
enqueue(&queue, 20);
enqueue(&queue, 30);
// Peek at the front element
printf("Front element: %d\n", peek(&queue));
// Dequeue elements from the queue
printf("Dequeued %d\n", dequeue(&queue));
printf("Dequeued %d\n", dequeue(&queue));
// Display the front element again
printf("Front element: %d\n", peek(&queue));
// Dequeue remaining elements
printf("Dequeued %d\n", dequeue(&queue));
// Attempt to dequeue from an empty queue
dequeue(&queue);
return 0;
}
```

Output:-