Experiment No. 7

Aim:

To implement a job queue to simulate the process of handling pizza orders in a pizza parlor, using the First Come First Served (FCFS) approach.

Objective:

- To understand the concept of the queue data structure and its operations.
- To simulate a real-world queue scenario such as order processing.
- To perform enqueue (add job), dequeue (delete job), and display operations.

Problem Statement:

Design and implement a program to simulate a job queue for a pizza parlor that can accept a maximum of M orders. Orders must be processed on a First Come First Served (FCFS) basis. The system should support the following:

- Adding a new job (order)
- Deleting the next job (order served)
- Displaying all pending jobs

Theory:

Queue is a linear structure which follows a particular order in which the operations are performed. The order is **F**irst **I**n **F**irst **O**ut (FIFO). A good example of queue is any queue of consumers for a resource where the consumer that came first is served first.
The difference between stacks and queues is in removing. In a stack we remove the item the most recently added; in a queue, we remove the item the least recently added.

Array implementation of Queue

For implementing queue, we need to keep track of two indices, front and rear. We enqueue an item at the rear and dequeue an item from front.

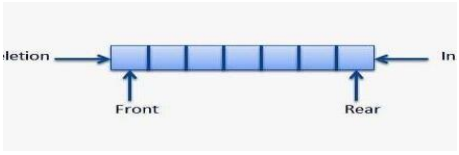

Figure: Representation of queue

Operations on Queue:

Mainly the following four basic operations are performed on queue:

- enqueue(): Adds an item to the queue. If the queue is full, then it is said to be an Overflow condition.
- dequeue(): Removes an item from the queue. The items are popped in the same order in which they are pushed. If the queue is empty, then it is said to be an Underflow condition.
- isfull(): checks if queue is full.
- isempty(): checks if queue is empty

isfull() Operation:	**isempty() Operation :**

bool isfull()
{
if(rear == MAXSIZE - 1)
return true;
else
return false;
}
	bool isempty()
	{
	if(rear==front-1)
	return true;
	else
	return false;
	}

Enqueue Operation(i.e. insertion):	**Dequeue Operation(i.e deletion):**
int enqueue(int data)	int dequeue()
if(isfull())	if(isempty())
return 0;	return 0;

rear = rear + 1;
queue[rear] = data;
return 1;
	int data = queue[front];
	front = front + 1;
	return data;

Algorithm:

1. Initialize

- Set front = -1, rear = -1
- Define queue array of maximum size MAX

2. Add Job (Enqueue)

If rear == MAX - 1: Queue is full (Overflow)

Else:

- If front == -1, set front = 0
- Increment rear

- Add job to queue[rear]

3. Delete Job (Dequeue)
   - If front == -1 or front > rear: Queue is empty (Underflow)
   - Else:
     - Process job at queue[front]
     - Increment front
     - If front > rear, reset front = rear = -1
4. Display Jobs

- If queue is empty, print appropriate message.
- Else, loop from front to rear and print each job.

Input:

User selects options from a menu:
- Add Job
- Delete Job
- Display Jobs
- Exit

Output:

Note: Write output of your program

Conclusion:

In this lab, we successfully implemented a **queue** using arrays in C to simulate a **pizza order management system**. The queue operates on a **First Come First Served** basis and demonstrates basic queue operations: **enqueue**, **dequeue**, and **display**. This simulation helps understand how queue data structures are used in real-world applications like job scheduling and order processing.