**THEORY:**

A queue is a fundamental data structure that follows the First In, First Out (FIFO) principle, meaning that the first element added to the queue will be the first one to be removed. It can be visualized as a line of people waiting for a service, where the person who arrives first is the first to be served. Queues are commonly used in various computing applications, such as process scheduling, task management, and network data packet handling.

The basic operations associated with a queue are as follows:

1. Enqueue (Insertion):

- Operation: Adds an element to the end (rear) of the queue.

- Description: The new element becomes the last in the queue, waiting for its turn to be processed.

2. Dequeue (Deletion):

- Operation: Removes the element from the front (head) of the queue.

- Description: The element that has been in the queue the longest is removed, indicating that it has been processed.

3. Front (Peek):

- Operation: Returns the element at the front of the queue without removing it.

- Description: Allows you to inspect the next element to be processed without actually removing it from the queue.

4. isEmpty:

- Operation: Checks if the queue is empty.

- Description: Returns a boolean value indicating whether the queue is empty or contains elements.

5. isFull (in bounded queues):

- Operation: Checks if the queue is full and cannot accept more elements.

- Description: Relevant in scenarios where the queue has a fixed size, and capacity needs to be monitored.

**PROBLEM:**

Write a program to perform the following operations:

1. Enqueue
2. Dequeue
3. Display all elements
4. Display front element

**PROBLEM ANALYSIS:**

The task involves designing a simple program for basic queue operations, including enqueue, dequeue, displaying all elements, and displaying the front element. The fundamental steps include handling user inputs for enqueue, managing the removal of elements from the front for dequeue, traversing the queue for displaying all elements, and presenting the front element without removal. Key considerations encompass dependency analysis, requirement identification, data and algorithmic choices, constraint handling, and a feasibility study for a straightforward and effective implementation. Success criteria involve error-free operations, graceful handling of edge cases, and simplicity in both data structures and algorithms.

**ALGORITHM:**

1. Initialize an empty queue using an array.

2. Repeat the following steps until the user decides to exit:

a. Display a menu of operations

b. Accept the user's choice.

c. Perform the corresponding operation based on the user's choice.

- For Enqueue:

i. Prompt the user for the element to be enqueued.

ii. Add the element to the end (rear) of the queue.

- For Dequeue:

i. Check if the queue is empty.

ii. If not empty, remove the element from the front of the queue.

iii. If empty, display an appropriate message.

- For Display All Elements:

i. Traverse the entire queue and display each element.

ii. If the queue is empty, display an appropriate message.

- For Display Front Element:

i. Display the element at the front of the queue without removing it.

ii. If the queue is empty, display an appropriate message.

- For Exit:

i. Display a farewell message and terminate the program.

d. Repeat the menu loop until the user chooses to exit.

3. End.