**Practical No. 9**

**Aim:** Write a program to implement insertion and deletion operations in a queue using linear array.

**Theory:**

A **queue** is a linear data structure that follows the **FIFO (First In First Out)** principle.

* **Insertion (Enqueue):** Adding an element at the **rear** of the queue.
* **Deletion (Dequeue):** Removing an element from the **front** of the queue.

In a **linear queue using an array**:

* We maintain two indices: front and rear.
* Initially, both are set to -1 (queue is empty).
* On insertion, rear is incremented and the new element is stored.
* On deletion, the element at front is removed and front is incremented.
* If rear reaches the maximum size and more elements cannot be added without shifting, it may cause the **queue overflow problem** (even if some space is free at the beginning). This is solved by **circular queues**, but here we use a simple linear queue.

**Algorithm:**

**Algorithm for Insertion (Enqueue):**

1. Check if rear == MAX - 1. If yes, print "Queue Overflow".
2. If front == -1, set front = 0.
3. Increment rear by 1.
4. Insert the new element at queue[rear].

**Algorithm for Deletion (Dequeue):**

1. Check if front == -1 or front > rear. If yes, print "Queue Underflow".
2. Print the deleted element at queue[front].
3. Increment front by 1.

**Program:**

#include <stdio.h> // For input and output functions

#define MAX 5 // Define maximum size of the queue

int queue[MAX]; // Array to store queue elements

int front = -1; // Front pointer

int rear = -1; // Rear pointer

// Function to insert element into queue

void enqueue(int value) {

if (rear == MAX - 1) { // If rear has reached end, queue is full

printf("Queue Overflow!\n");

} else {

if (front == -1) // If inserting first element

front = 0;

rear++; // Increment rear

queue[rear] = value; // Insert element at rear

printf("Inserted %d\n", value); // Print confirmation

}

}

// Function to delete element from queue

void dequeue() {

if (front == -1 || front > rear) { // If queue is empty

printf("Queue Underflow!\n");

} else {

printf("Deleted %d\n", queue[front]); // Print deleted element

front++; // Increment front to remove element

}

}

// Function to display the queue

void display() {

if (front == -1 || front > rear) { // If queue is empty

printf("Queue is empty\n");

} else {

printf("Queue elements: ");

for (int i = front; i <= rear; i++) { // Print elements from front to rear

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

}

printf("\n");

}

}

int main() {

// Demonstration of queue operations

enqueue(10); // Insert 10

enqueue(20); // Insert 20

enqueue(30); // Insert 30

display(); // Show current queue

dequeue(); // Delete element

display(); // Show updated queue

enqueue(40); // Insert 40

enqueue(50); // Insert 50

enqueue(60); // Try inserting when full

display(); // Final queue state

return 0; // End of program

}

**Output (Sample Run):**

Inserted 10

Inserted 20

Inserted 30

Queue elements: 10 20 30

Deleted 10

Queue elements: 20 30

Inserted 40

Inserted 50

Queue Overflow!

Queue elements: 20 30 40 50

**Conclusion:**

* Successfully implemented a **linear queue using arrays**.
* Learned how to perform **insertion (enqueue)** and **deletion (dequeue)** operations.
* Understood limitations of a linear queue, such as **overflow when rear reaches MAX-1**, even if there is space at the front.
* This provides the foundation for understanding **circular queues** which overcome this limitation.