LAB SESSION 06

Queue Implementation Using Arrays

THEORY

A queue is a linear data structure that follows the First In First Out (FIFO) principle — the element that is inserted first is removed first.
Think of a queue in the real world, like a line at a bank counter:

- The person who joins first gets served first.
- New arrivals go to the end of the line.

Queue Characteristics:

- Enqueue: Insert an element at the rear (end) of the queue.
- Dequeue: Remove an element from the front (start) of the queue.
- Front: The position from which elements are removed.
- Rear: The position where elements are added.

Queue Using Arrays
In an **array-based queue**, a fixed-size array is used along with two variables:

- **front** → index of the first element
- **rear** → index of the last element

Initially:
front = -1; rear = -1;
Limitations of Basic Array Queue
When elements are removed, the unused space at the beginning of the array cannot be reused unless circular queue logic is applied.
Applications

- Task scheduling
- Printer queue management
- Call center systems
- Breadth-first search (BFS) in graphs

PROCEDURE

1. Declare a fixed-size array and initialize front and rear to -1.
2. Implement **enqueue** to insert at the rear.
3. Implement **dequeue** to remove from the front.
4. Implement **display** to show the queue contents.
5. Add checks for queue overflow and underflow.
6. Test with sample inputs.

CODE

```
#include <iostream>
#define SIZE 5
using namespace std;

int queueArr[SIZE];
```

```
int front = -1, rear = -1;

void enqueue(int value) {
   if (rear == SIZE - 1) {
      cout << "Queue Overflow!\n";
      return;
   }
   if (front == -1) front = 0; // First element
   queueArr[++rear] = value;
   cout << value << " enqueued into queue.\n";
}

void dequeue() {
   if (front == -1 || front > rear) {
      cout << "Queue Underflow!\n";
      return;
   }
   cout << queueArr[front++] << " dequeued from queue.\n";
}

void display() {
   if (front == -1 || front > rear) {
      cout << "Queue is empty.\n";
      return;
   }
   cout << "Queue elements: ";
   for (int i = front; i <= rear; i++)
      cout << queueArr[i] << " ";
   cout << endl;
}

int main() {
   enqueue(10);
   enqueue(20);
   enqueue(30);
   display();
   dequeue();
   display();
   return 0;
}
```


EXPECTED OUTPUT

```
10 enqueued into queue.
20 enqueued into queue.
30 enqueued into queue.
Queue elements: 10 20 30
10 dequeued from queue.
Queue elements: 20 30
```

EXERCISE

1. Modify the code to allow user input for enqueue and dequeue operations.
2. Implement a **peek()** function to return the front element without removal.
3. Handle **circular queue** implementation using arrays.
4. Write a function to check if the queue is full or empty.
5. Implement queue using a linked list for dynamic memory allocation.