# ✅ SESSION 5: QUEUES

---

## 🧠 1. What is a Queue?

A **queue** is a **linear data structure** used to store elements in an **ordered way**, following the principle:

### 🔁 FIFO – First In, First Out

- The **first element inserted** into the queue will be the **first one to be removed**.
- Just like standing in a line: the person who comes **first** is served **first**.

---

## 🎯 2. Real-Life Analogy: Ticket Counter

Imagine a line at a **movie ticket counter**:

- People enter the line **from the end** (rear).
- Tickets are given **from the front**.
- The person who joined first gets the ticket first.

This is exactly how **queues** work:

```
Front →      [Person1, Person2, Person3]      ← Rear
        (First Out)                 (Last In)
```

---

## 🧾 3. Queue Terminology

| Term | Description |
|---|---|
| **Front** | Points to the first element of the queue. It is the **removal point**. |
| **Rear** | Points to the last element where a new element will be **inserted**. |
| **Enqueue** | **Insert** operation – adds an element at the rear. |

| | |
|---|---|
| **Dequeue** | **Remove** operation – removes an element from the front. |
| **Overflow** | Condition when the queue is full (cannot insert). |
| **Underflow** | Condition when the queue is empty (cannot remove). |

---

## 📚 4. Queue Rules and Behavior

- Queue uses **sequential memory** (in arrays).
- Operations are **non-reversible** like stack:
    - Stack → LIFO (Last In, First Out)
- **Fixed size** in static array: must handle full/empty conditions carefully.

---

## 🧩 5. Queue Using Array – C++ Implementation (with Functions)

### ✅ Code Explanation:

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

// Global Queue and Pointers
int queue[SIZE];
int front = -1, rear = -1;

// Enqueue: Insert element
void enqueue(int value) {
    if (rear == SIZE - 1) {
        cout << "Queue Overflow! Cannot insert " << value << endl;
        return;
    }
    if (front == -1) front = 0; // First insertion
    rear++;
    queue[rear] = value;
    cout << value << " inserted at position " << rear << endl;
```

```
}

// Dequeue: Remove element
void dequeue() {
    if (front == -1 || front > rear) {
        cout << "Queue Underflow! No elements to remove." << endl;
        return;
    }
     cout << queue[front] << " removed from position " << front <<
endl;
    front++;
    if (front > rear) {
        // Reset queue when all elements removed
        front = rear = -1;
    }
}

// Display queue elements
void display() {
    if (front == -1 || front > rear) {
        cout << "Queue is Empty." << endl;
        return;
    }
    cout << "Queue Elements: ";
    for (int i = front; i <= rear; i++) {
        cout << queue[i] << " ";
    }
    cout << endl;
}

// Main Function: Menu using while loop and switch-case
int main() {
    int choice, value;

    cout << "Queue Implementation using Array (Menu-Based)\n";

    while (true) {
        // Menu
        cout << "\n=== MENU ===\n";
        cout << "1. Enqueue (Insert)\n";
        cout << "2. Dequeue (Remove)\n";
```

```
        cout << "3. Display Queue\n";
        cout << "4. Exit\n";
        cout << "Enter your choice (1-4): ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                enqueue(value);
                break;

            case 2:
                dequeue();
                break;

            case 3:
                display();
                break;

            case 4:
                cout << "Exiting program. Thank you!\n";
                return 0; // End the program

            default:
                    cout << "Invalid choice. Please enter a number
between 1 and 4.\n";
        }
    }

    return 0;
}
```

---

# ✅ Queue Quiz

**📘 Topic: Queue using Array + Menu-based Program**

**🎯 Objective: To test students' understanding of FIFO, terminology, overflow/underflow, array implementation, dry run, and logic.**

---

## 🧪 PART A: Theoretical Questions (1 mark each)

1. What does FIFO stand for in the context of a queue?
2. What is the difference between a stack and a queue?
3. Define the terms**:**
   **a)** `enqueue`
   **b)** `dequeue`
   **c)** `front`
   **d)** `rear`
4. When does **Queue Overflow** occur?
5. When does **Queue Underflow** occur?
6. What is the initial value of `front` and `rear` in an empty queue?
7. After inserting 3 elements, what are the values of `front` and `rear`?
8. What will happen if you keep inserting elements without deleting in a static array queue?
9. What is the condition to check if a queue is **empty**?
10. What is the condition to check if a queue is **full**?

---

## 🧮 PART B: Dry Run – Trace the Output (2 marks each)

### Q11. What will be the output of the following?

```
enqueue(10);
enqueue(20);
dequeue();
enqueue(30);
display();
```

---

### Q12. If front = 1 and rear = 3, and the queue array is:

```
[10, 20, 30, 40, _]
```

What elements will be displayed?

---

### Q13. Perform a dry run for this sequence:

```
enqueue(5);
enqueue(8);
enqueue(9);
enqueue(12);
enqueue(15);
enqueue(20); // Note: Queue size is 5
```

What will be the final output and message?

---

## 🧠 PART C: Code Understanding (2 marks each)

### Q14. What is the purpose of this line?

```
if (front > rear) front = rear = -1;
```

---

### Q15. Why do we use the condition:

```
if (front == -1 || front > rear)
```

in both `dequeue()` and `display()` functions?

---

### Q16. In a menu-based queue, what does this line do?

```
cin >> choice;
```

---

## ✏️ PART D: Predict the Output (2 marks each)

### Q17. What will be printed?

```
enqueue(100);
enqueue(200);
dequeue();
dequeue();
dequeue();
```

---

### Q18. What happens when we try to `dequeue()` from an empty queue?

---

## ✅ BONUS Question (5 marks)

**Q19. Write a small program segment (just 3-4 lines) that shows how to insert an element into a queue using `enqueue()` and then display the queue.**

---

## 🎯 Answer Key

### ✅ PART A:

1. First In First Out
2. Stack = LIFO, Queue = FIFO

3. a) insert b) remove c) front index d) rear index
4. When rear reaches SIZE - 1
5. When front is -1 or front > rear
6. -1 and -1
7. front = 0, rear = 2
8. Overflow
9. front == -1 or front > rear
10. rear == SIZE - 1

## ✅ PART B:

11. Output:

```
10 inserted
20 inserted
10 removed
30 inserted
Queue Elements: 20 30
```

12. Output: **20 30 40**
13. Output:

```
5 inserted
8 inserted
9 inserted
12 inserted
15 inserted
Queue Overflow! Cannot insert 20
```

## ✅ PART C:

14. Resets the queue to empty when all elements are removed
15. To check if the queue is empty

16. Takes user input for menu option

## ✅ PART D:

17.

```
100 inserted
200 inserted
100 removed
200 removed
Queue Underflow! No elements to remove.
```

18. Underflow error message