Circular Queue and Deque in C++

Part 1️⃣: 🔄 Circular Queue

✅ 1. Definition

- A **Circular Queue** is a **linear data structure** that follows the **FIFO** principle (First-In-First-Out).
- Unlike a normal queue, it **wraps around** when the end of the array is reached.
- This helps to **reuse the empty spaces** left after deletions.

✅ 2. Real-Life Example

Imagine people standing in a **circular line** for a roller coaster ride — when someone leaves the front, the space can be reused from the rear without shifting everyone.

✅ 3. Circular Queue Structure

```
#define SIZE 5
int queue[SIZE];
int front = -1, rear = -1;
```

- `front`: Points to the first element
- `rear`: Points to the last inserted element

✅ 4. Full Conditions

- **Empty:** `front == -1`
- **Full:** `(rear + 1) % SIZE == front`

✅ 5. Operations in Circular Queue

🔹 **Enqueue (Insert)**

```
void enqueue(int value) {
    if ((rear + 1) % SIZE == front)
        cout << "Queue is Full\n";
    else {
        if (front == -1)
            front = rear = 0;
        else
            rear = (rear + 1) % SIZE;
        queue[rear] = value;
    }
}
```

🔹 **Dequeue (Delete)**

```
void dequeue() {
    if (front == -1)
        cout << "Queue is Empty\n";
    else {
        cout << "Deleted: " << queue[front] << endl;
        if (front == rear)
            front = rear = -1;
        else
            front = (front + 1) % SIZE;
    }
}
```

🔹 **Display Queue**

```
void display() {
    if (front == -1)
        cout << "Queue is Empty\n";
    else {
        int i = front;
        cout << "Queue: ";
        while (true) {
            cout << queue[i] << " ";
            if (i == rear) break;
            i = (i + 1) % SIZE;
```

```
        }
        cout << endl;
    }
}
```

🧠 Dry Run Example

Operations:

```
enqueue(10); enqueue(20); enqueue(30); enqueue(40); enqueue(50);  // Full
dequeue(); dequeue();
enqueue(60); enqueue(70);  // Wraps around
display();
```

Result:

```
Queue: 30 40 50 60 70
```

🧪 Summary of Circular Queue

Operation	Time Complexity
Enqueue	O(1)
Dequeue	O(1)
Display	O(n)
Space Used	Fixed, reused

Part 2️⃣: 🔁 Deque (Double-Ended Queue)

✅ 1. Definition

- A **Deque** is a **double-ended queue** where we can insert and delete from both **front** and **rear**.
- It is more flexible than both a queue and a stack.

✅ 2. Real-Life Analogy

Think of a **bus** with doors at both front and back. Passengers can **enter and exit from either side**.

✅ 3. Types of Deques

Type	Insertion	Deletion
Input Restricted	Only at rear	Both front and rear
Output Restricted	Both front and rear	Only at front
General Deque	Both ends	Both ends

✅ 4. Deque Structure

```
#define SIZE 5
int deque[SIZE];
int front = -1, rear = -1;
```

- `front` and `rear` are used for both insertion and deletion.

✅ 5. Conditions

- **Empty:** `front == -1`
- **Full:** `(front == (rear + 1) % SIZE)`

✅ 6. Operations in Deque

➕ Insert at Front

```
void insertFront(int value) {
    if ((front == (rear + 1) % SIZE))
        cout << "Deque is Full\n";
    else {
        if (front == -1)
            front = rear = 0;
        else
            front = (front - 1 + SIZE) % SIZE;
        deque[front] = value;
    }
}
```

➕ Insert at Rear

```
void insertRear(int value) {
    if ((front == (rear + 1) % SIZE))
        cout << "Deque is Full\n";
    else {
        if (front == -1)
            front = rear = 0;
        else
            rear = (rear + 1) % SIZE;
        deque[rear] = value;
    }
}
```

➖ Delete from Front

```
void deleteFront() {
    if (front == -1)
        cout << "Deque is Empty\n";
    else {
        cout << "Deleted: " << deque[front] << endl;
        if (front == rear)
            front = rear = -1;
        else
```

```
            front = (front + 1) % SIZE;
        }
}
```

➖ Delete from Rear

cpp
CopyEdit

```cpp
void deleteRear() {
    if (front == -1)
        cout << "Deque is Empty\n";
    else {
        cout << "Deleted: " << deque[rear] << endl;
        if (front == rear)
            front = rear = -1;
        else
            rear = (rear - 1 + SIZE) % SIZE;
    }
}
```

📟 Display

```cpp
void display() {
    if (front == -1)
        cout << "Deque is Empty\n";
    else {
        int i = front;
        cout << "Deque: ";
        while (true) {
            cout << deque[i] << " ";
            if (i == rear) break;
            i = (i + 1) % SIZE;
        }
        cout << endl;
    }
}
```

🧠 Dry Run Example

Operations:

```cpp
insertRear(10);
```

```
insertRear(20);
insertFront(5);
deleteRear();
deleteFront();
display();
```

Deque Status:

```
Deque: 10
```

🧪 Summary of Deque

Operation	Description	Time Complexity
insertFront()	Insert element at front	O(1)
insertRear()	Insert element at rear	O(1)
deleteFront()	Delete from front	O(1)
deleteRear()	Delete from rear	O(1)

📝 Quiz (Quick Recap)

Q1. What is the condition for a circular queue to be full?
✅ `(rear + 1) % SIZE == front`

Q2. What is the advantage of a circular queue over a normal queue?
✅ Space is reused after deletion

Q3. What does a deque allow?
✅ Insertion and deletion from both ends

Q4. In input-restricted deque, from where can we delete?
✅ From both front and rear