📘 Teaching Notes: Types of Queues (Unit 2 – Part 2)

---

4. Types of Queues

4.1 Circular Queue

Concept and Need:

- Solves the unused space problem of linear queue.
- Rear connects to front to form a circle.

Array Implementation & Operations:

```c
#include <stdio.h>
#define MAX 5

int queue[MAX];
int front = -1, rear = -1;

int isFull() {
    return ((rear + 1) % MAX == front);
}

int isEmpty() {
    return (front == -1);
}

void enqueue(int value) {
    if(isFull()) { printf("Circular Queue Overflow\n"); return; }
    if(front == -1) front = 0;
    rear = (rear + 1) % MAX;
    queue[rear] = value;
    printf("%d enqueued\n", value);
}

void dequeue() {
    if(isEmpty()) { printf("Circular Queue Underflow\n"); return; }
    printf("%d dequeued\n", queue[front]);
    if(front == rear) front = rear = -1;
    else front = (front + 1) % MAX;
}

void display() {
    if(isEmpty()) { printf("Queue is empty\n"); return; }
    printf("Queue elements: ");
    int i = front;
    while(1) {
        printf("%d ", queue[i]);
```

```
        if(i == rear) break;
        i = (i + 1) % MAX;
    }
    printf("\n");
}

int main() {
    enqueue(10); enqueue(20); enqueue(30); enqueue(40);
    display();
    dequeue(); dequeue();
    display();
    enqueue(50); enqueue(60);
    display();
    return 0;
}
```

Advantages over Linear Queue: Efficient memory usage, avoids overflow if space is available at front.

4.2 Priority Queue

Introduction: Elements have priorities; higher priority served first.

Array Implementation (Descending Priority):

```
#include <stdio.h>
#define MAX 5

int queue[MAX];
int n = 0;

void enqueue(int value) {
    if(n == MAX) { printf("Priority Queue Overflow\n"); return; }
    int i = n - 1;
    while(i >= 0 && queue[i] < value) {
        queue[i+1] = queue[i];
        i--;
    }
    queue[i+1] = value;
    n++;
    printf("%d enqueued\n", value);
}

void dequeue() {
    if(n == 0) { printf("Priority Queue Underflow\n"); return; }
    printf("%d dequeued\n", queue[0]);
    for(int i=1;i<n;i++) queue[i-1] = queue[i];
    n--;
}
```

```c
void display() {
    if(n==0) { printf("Queue is empty\n"); return; }
    printf("Queue elements: ");
    for(int i=0;i<n;i++) printf("%d ",queue[i]);
    printf("\n");
}

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

Applications: CPU scheduling, task management, emergency systems.

4.3 Double Ended Queue (Deque)

Definition: Insertion and deletion possible at both ends.

Array Implementation:

```c
#include <stdio.h>
#define MAX 5

int deque[MAX];
int front=-1,rear=-1;

int isFull(){ return (rear==MAX-1 && front==0); }
int isEmpty(){ return (front==-1); }

void insertRear(int value){
    if(isFull()){ printf("Deque Overflow\n"); return; }
    if(front==-1){ front=rear=0; }
    else if(rear==MAX-1){ printf("Cannot insert at rear\n"); return; }
    else rear++;
    deque[rear]=value;
    printf("%d inserted at rear\n", value);
}

void insertFront(int value){
    if(isFull() || front==0){ printf("Cannot insert at front\n"); return; }
    if(front==-1){ front=rear=0; }
    else front--;
    deque[front]=value;
    printf("%d inserted at front\n", value);
}

void deleteFront(){
```

```
    if(isEmpty()){ printf("Deque Underflow\n"); return; }
    printf("%d deleted from front\n", deque[front]);
    if(front==rear) front=rear=-1;
    else front++;
}

void deleteRear(){
    if(isEmpty()){ printf("Deque Underflow\n"); return; }
    printf("%d deleted from rear\n", deque[rear]);
    if(front==rear) front=rear=-1;
    else rear--;
}

int main(){
    insertRear(10); insertRear(20); insertFront(5);
    deleteRear(); deleteFront();
    return 0;
}
```

4.4 Multiple Queues

Concept: Multiple queues managed simultaneously.

Applications: CPU scheduling, printer queue management.

Implementation Approaches:

- Array of queues
- Linked list of queues
- Each queue can operate independently with enqueue/dequeue functions.