

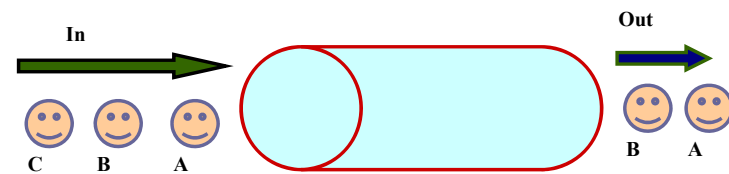
Stack and Queue

Part-II

1

Queue

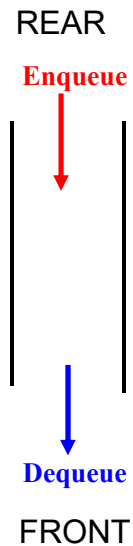
Data structure with **First-In First-Out (FIFO)** behavior



2

Typical Operations on Queue

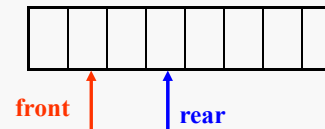
- isempty:** determines if the queue is empty
- isfull:** determines if the queue is full in case of a bounded size queue
- front:** returns the element at front of the queue
- enqueue:** inserts an element at the rear
- dequeue:** removes the element in front



3

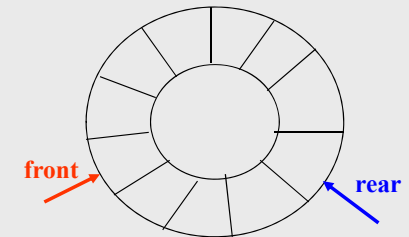
Possible Implementations

Linear Arrays:
(static/dynamically allocated)



Linked Lists: Use a linear linked list with `insert_rear` and `delete_front` operations

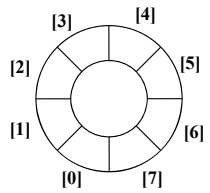
Circular Arrays:
(static/dynamically allocated)



Can be implemented by a 1-d array using modulus operations

4

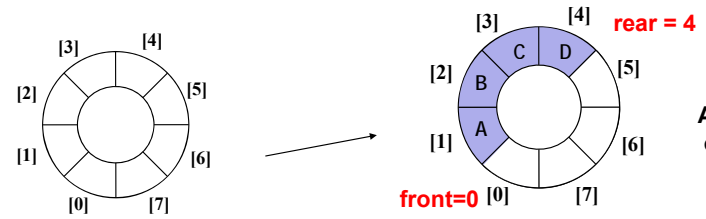
Circular Queue



front=0
rear=0

5

Circular Queue

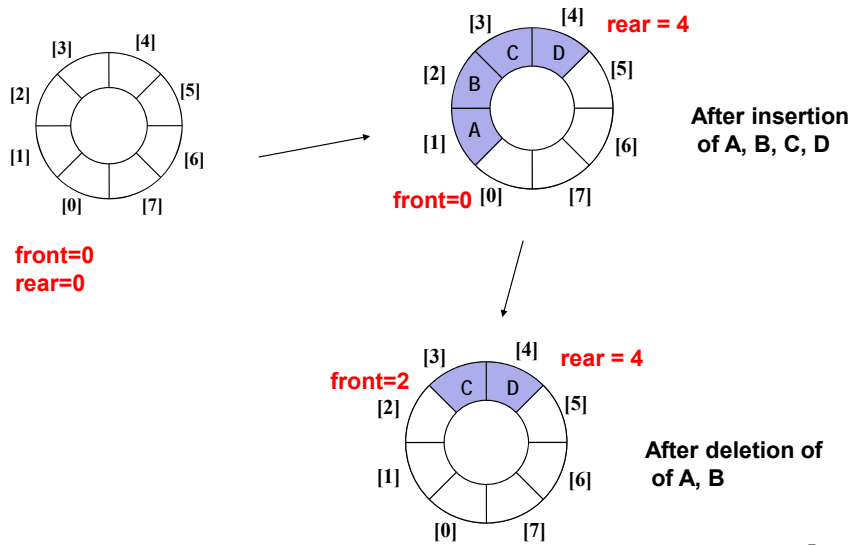


front=0
rear=0

After insertion
of A, B, C, D

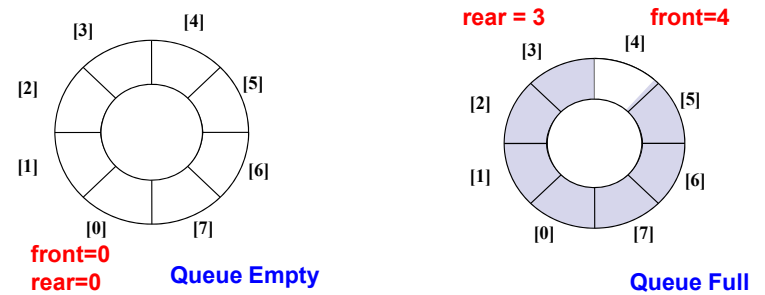
6

Circular Queue



7

front: index of queue-head (always empty)
rear: index of last element, unless rear = front



Queue Empty Condition: $front == rear$
Queue Full Condition: $front == (rear + 1) \% MAX_Q_SIZE$

8

Creating and Initializing a Circular Queue

Declaration

```
#define MAX_Q_SIZE 100
typedef struct {
    int key; /* just an example, can have
             any type of fields depending
             on what is to be stored */
} element;
typedef struct {
    element list[MAX_Q_SIZE];
    int front, rear;
} queue;
```

Create and Initialize

```
queue Q;
Q.front = 0;
Q.rear = 0;
```

9

Operations

```
int isfull (queue *q)
{
    if (q->front == ((q->rear + 1) %
                    MAX_Q_SIZE))
        return 1;
    return 0;
}
```

```
int isempty (queue *q)
{
    if (q->front == q->rear)
        return 1;
    return 0;
}
```

10

Operations

```
element front( queue *q )
{
    return q->list[(q->front + 1) % MAX_Q_SIZE];
}
```

```
void enqueue( queue *q, element e)
{
    q->rear = (q->rear + 1)%
              MAX_Q_SIZE;
    q->list[q->rear] = e;
}
```

```
void dequeue( queue *q )
{
    q-> front =
        (q-> front + 1)%
        MAX_Q_SIZE;
}
```

11

Practice Problems

- Implement the Queue as a linked list.
- Implement a **Priority Queue** which maintains the items in an order (ascending/ descending) and has additional functions like **remove_max** and **remove_min**
- Maintain a Doctor's appointment list

12