CS10003: Programming & Data Structures

Dept. of Computer Science & Engineering
Indian Institute of Technology Kharagpur

Autumn 2020

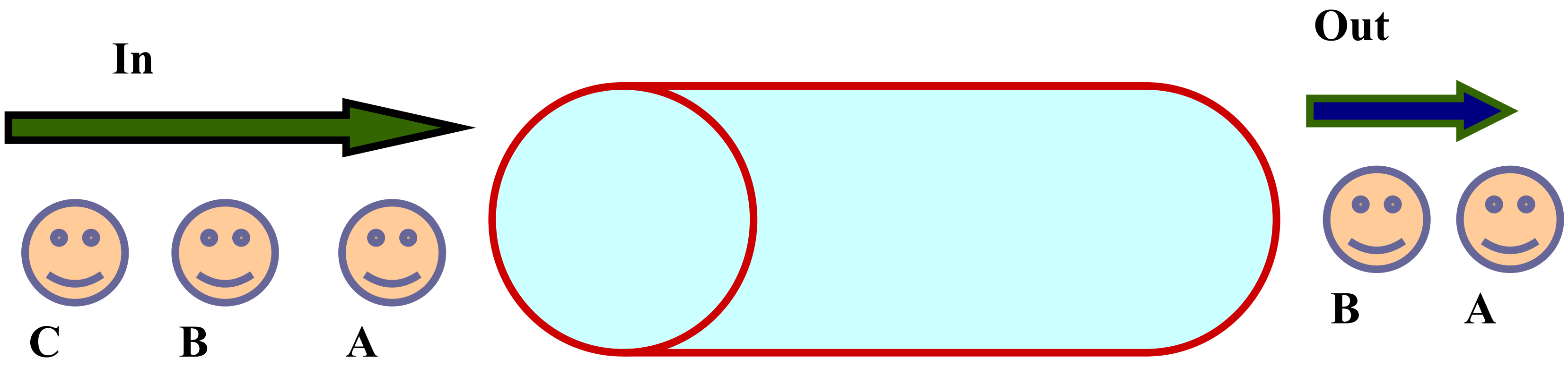
Queue

Queue

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

Queue

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



Typical Operations on Queue

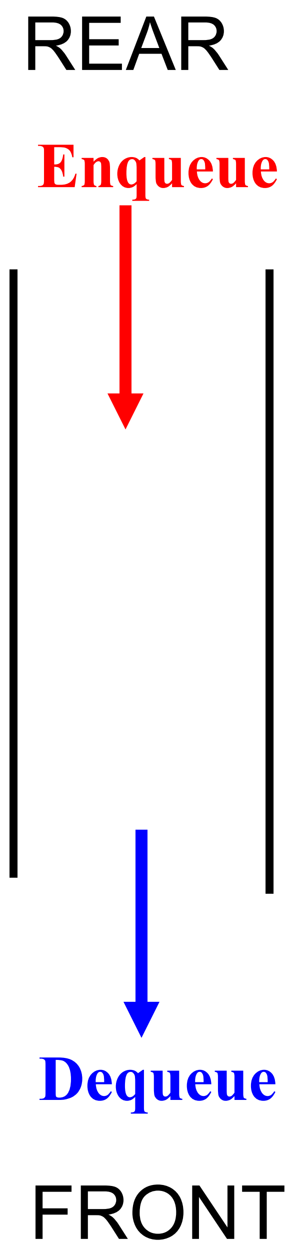
isempty: determines if the queue is empty

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

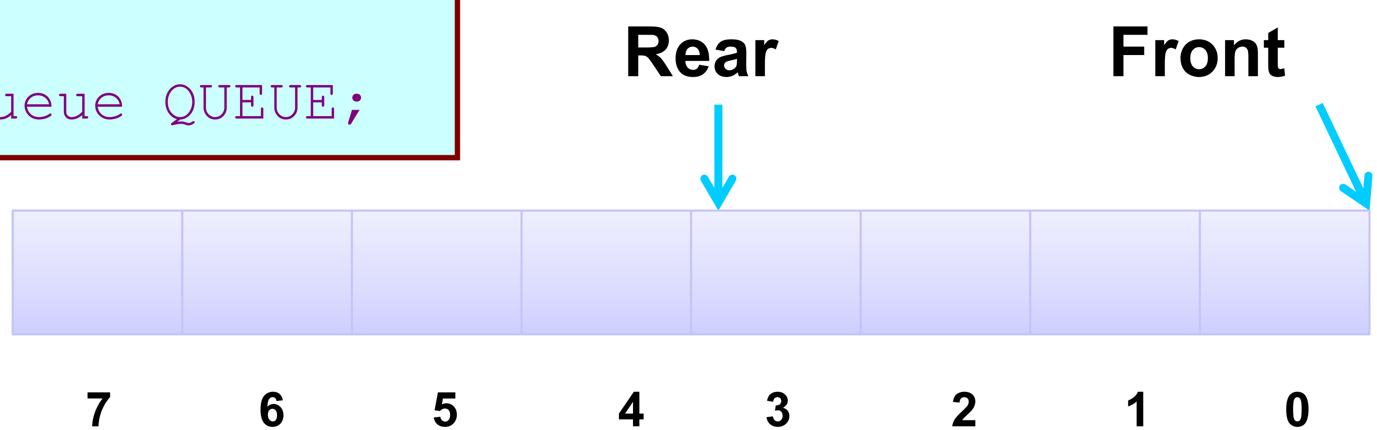


Queue using array

What do we need?

1. An array to store the elements (of maximum size).
2. Two integer variables (array index) to indicate front and rear.

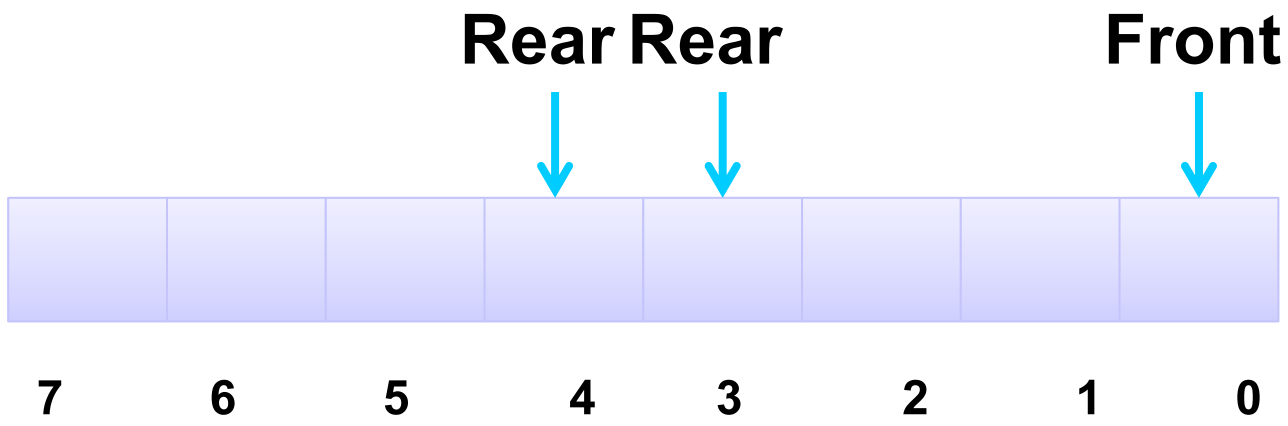
```
#define MAXSIZE 100
struct queue
{
   int que[MAXSIZE];
   int front,rear;
};
typedef struct queue QUEUE;
```



ENQUEUE

Increment rear (array index)

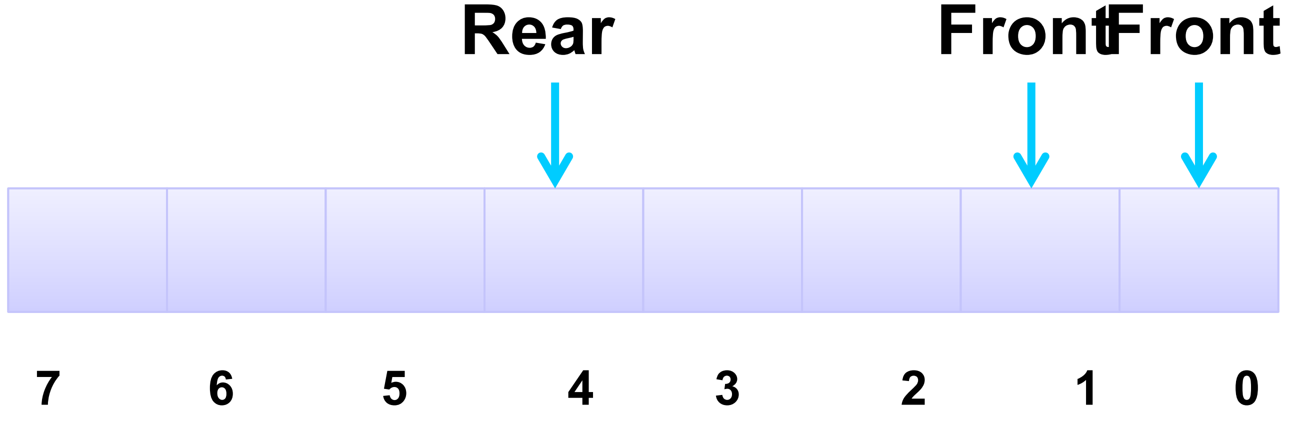
```
#define MAXSIZE 100
struct queue
{
    int que[MAXSIZE];
    int front,rear;
};
typedef struct queue QUEUE;
```



DEQUEUE

Increment front (array index)

```
#define MAXSIZE 100
struct queue
{
    int que[MAXSIZE];
    int front,rear;
};
typedef struct queue QUEUE;
```

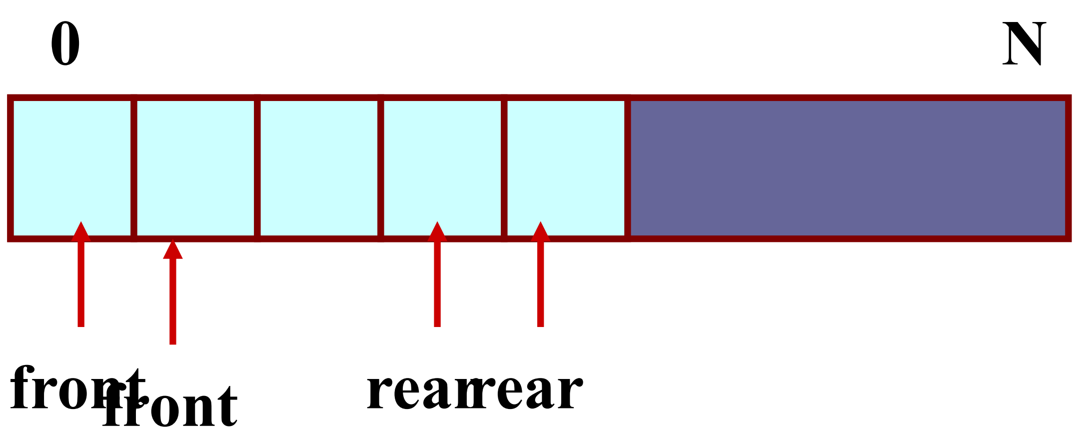


Problem with Array implementation

- The size of the queue depends on the number and order of enqueue and dequeue.
- It may be situation where memory is available but enqueue is not possible.

ENQUEUE **DEQUEUE**

Effective queuing storage area of array gets reduced.

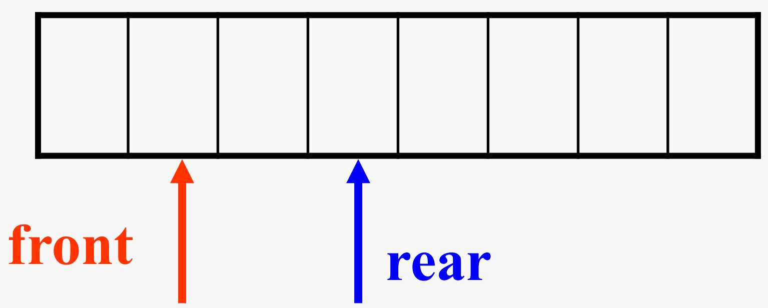


Use of circular array indexing

Possible Implementations

Linear Arrays:

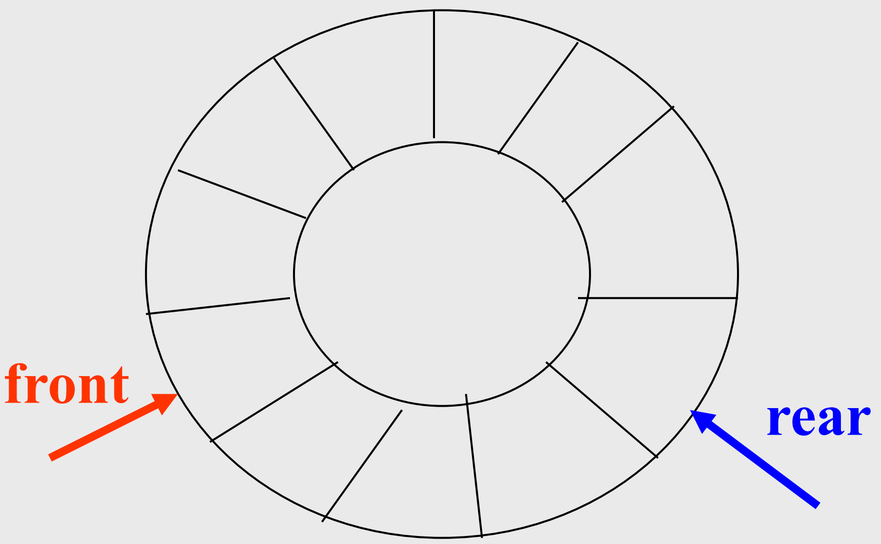
(static/dynamicaly allocated)



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

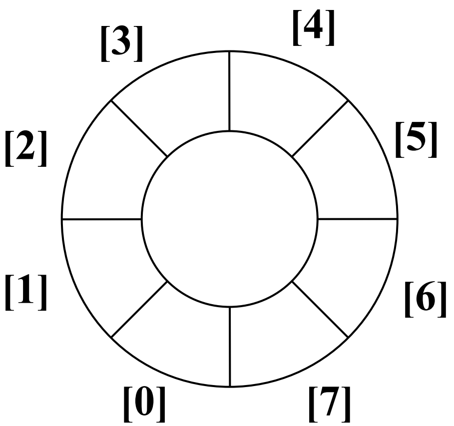
Circular Arrays:

(static/dynamically allocated)



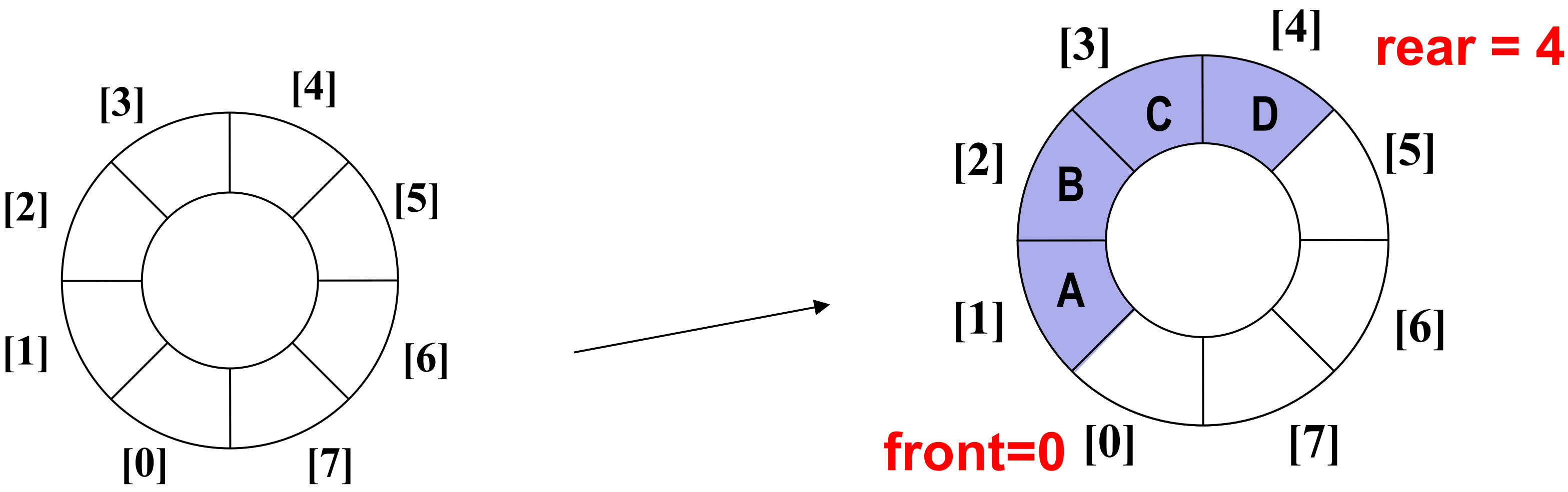
Can be implemented by a 1-d array using modulus operations

Circular Queue



front=0
rear=0

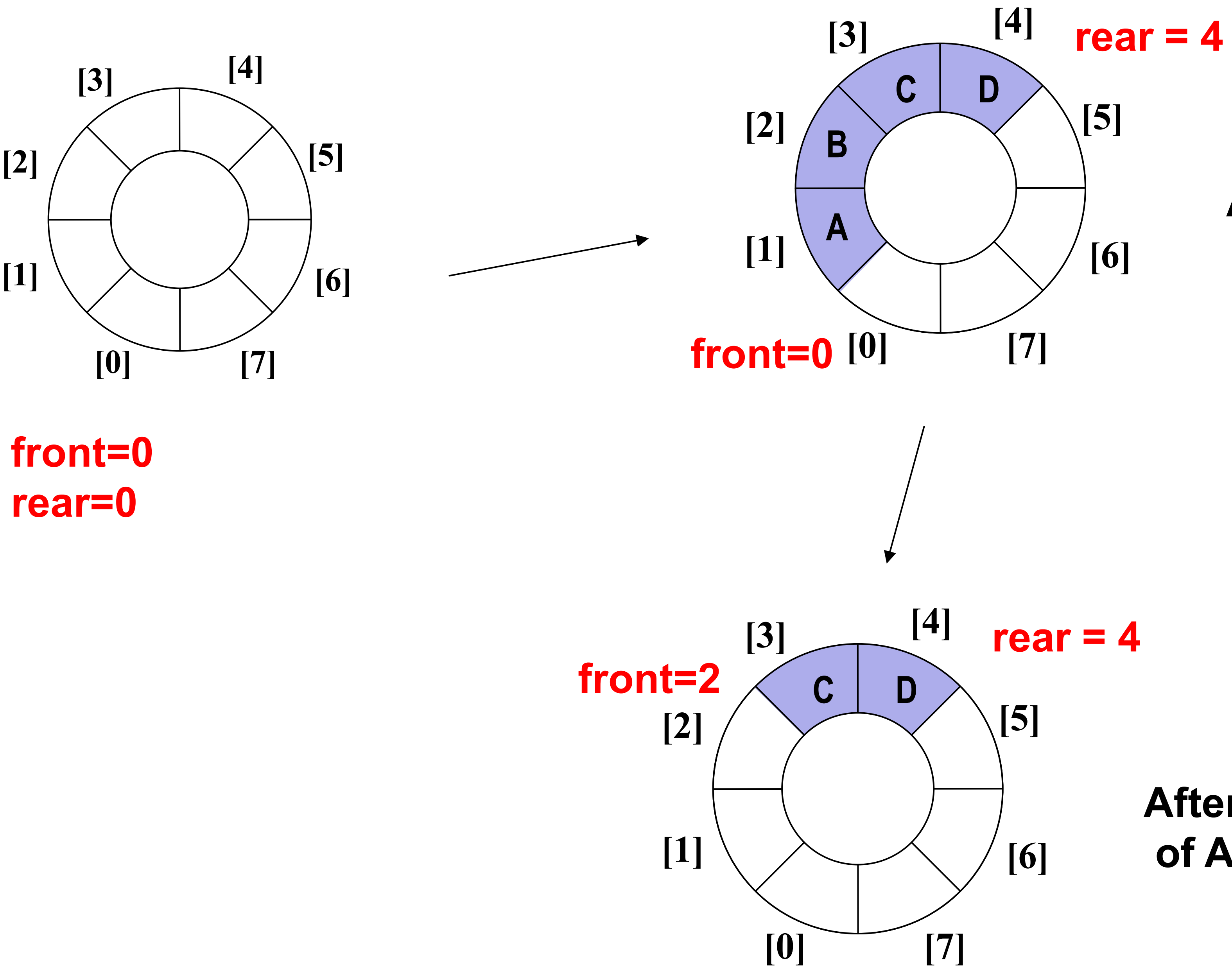
Circular Queue



After insertion of A, B, C, D

front=0
rear=0

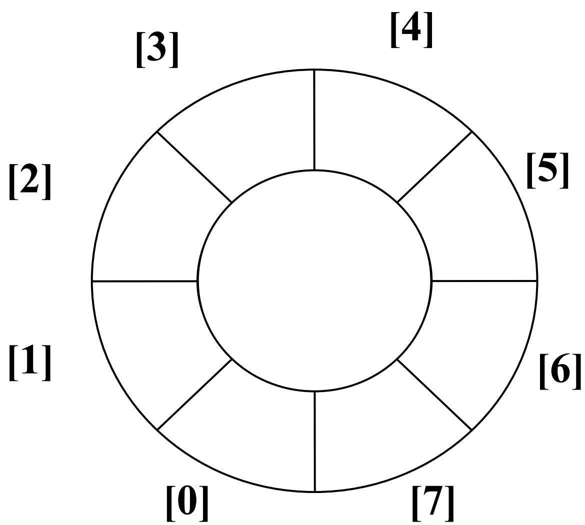
Circular Queue



After insertion of A, B, C, D

After deletion of of A, B

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

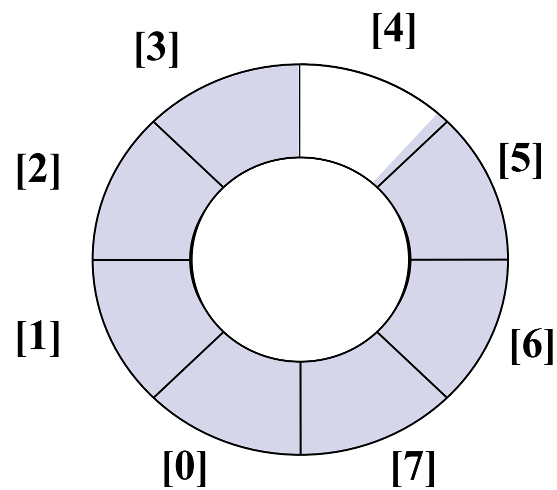


front=0
rear=0

Queue Empty

rear = 3

front=4



Queue Full

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

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;
```

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;
}
```

Operations

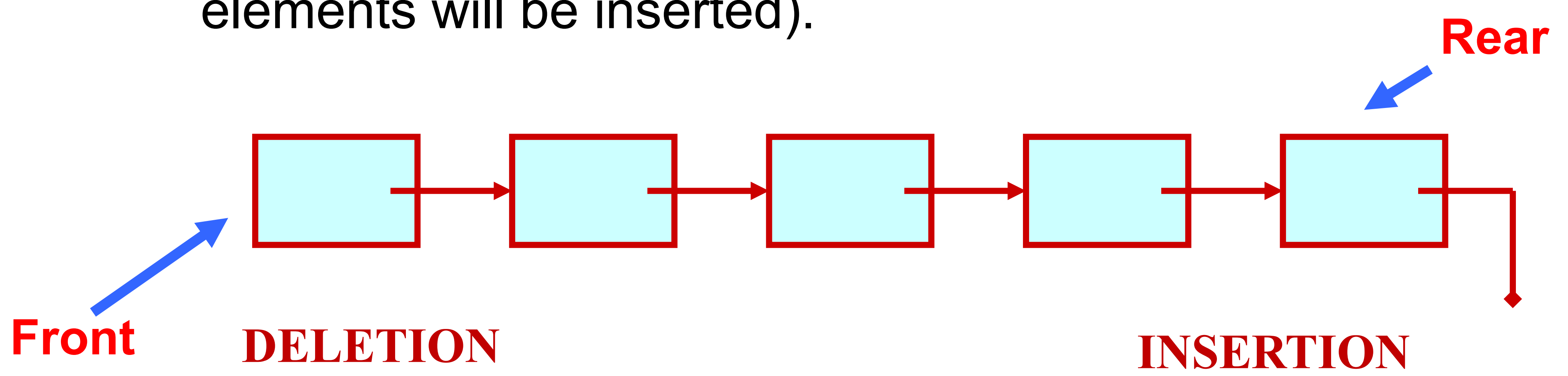
```
element front( queue *q )
{
    return q->list[(q->front + 1) % MAX_Q_SIZE];
}
```

```
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;
}
```

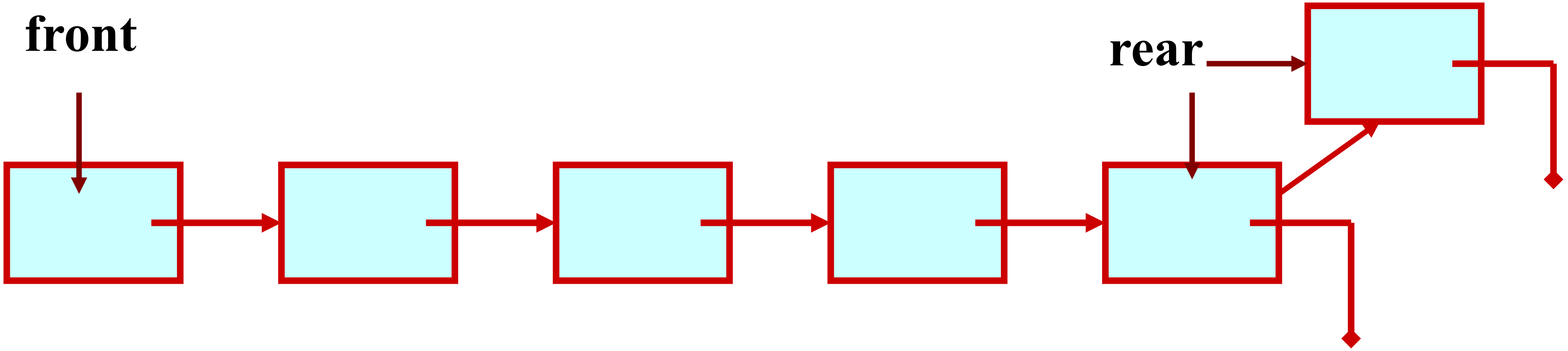
Queue using linked list

- Create a linked list to which items would be added to one end and deleted from the other end.
- Two pointers will be maintained:
  - One pointing to the beginning of the list (point from where elements will be deleted).
  - Another pointing to the end of the list (point where new elements will be inserted).



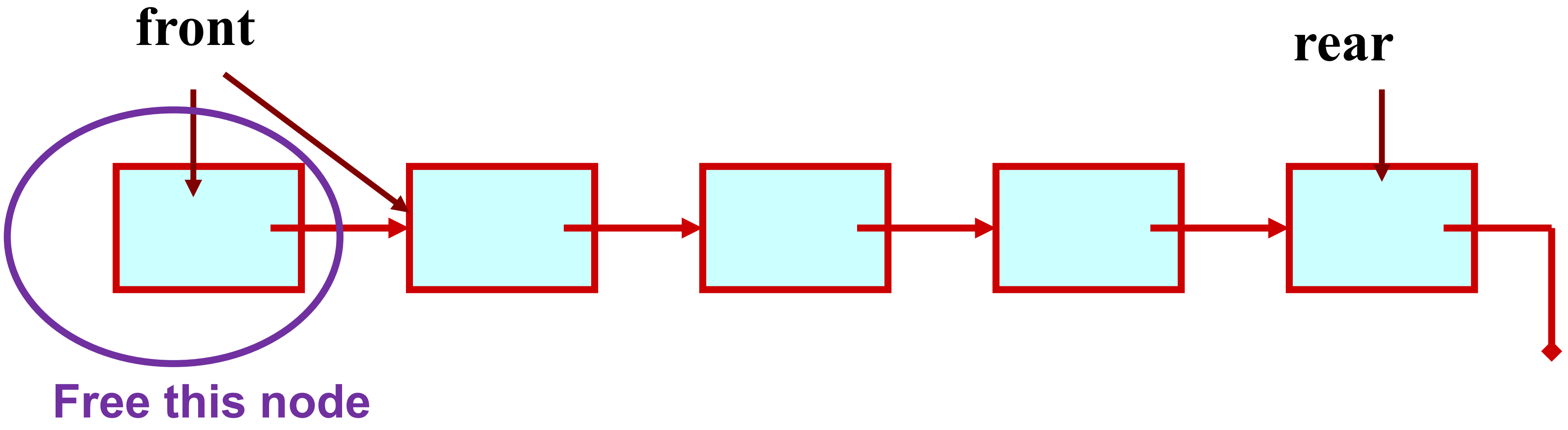
QUEUE: Insertion into a Linked List

ENQUEUE



QUEUE: Deletion from a Linked List

DEQUEUE

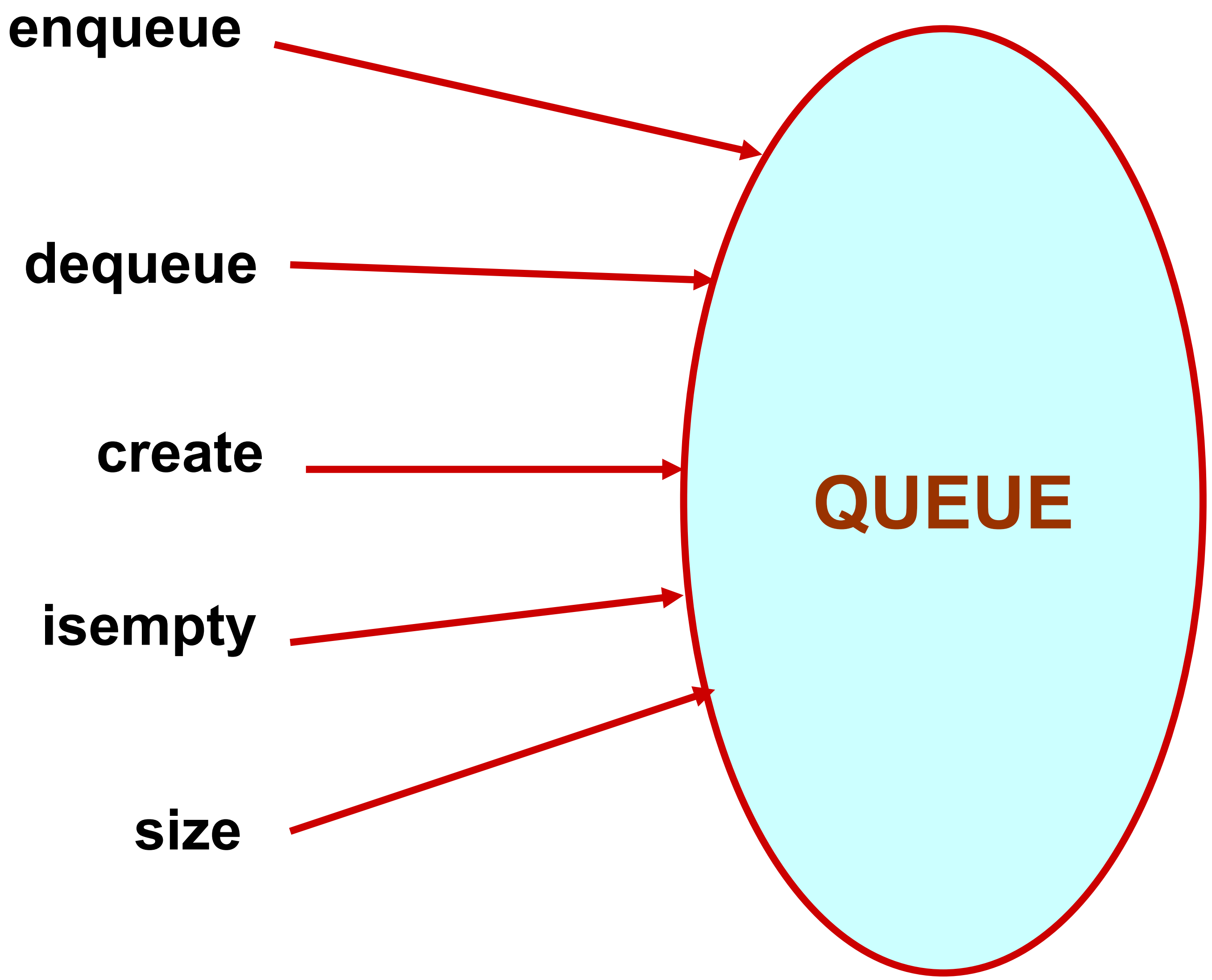


QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
void enqueue (QUEUE *q, int element);
                    /* Insert an element in the queue */
int dequeue (QUEUE *q);
                    /* Remove an element from the queue */
queue *create();
                    /* Create a new queue */
int isempty (QUEUE *q);
                    /* Check if queue is empty */
int size (QUEUE *q);
                    /* Return the no. of elements in queue */
int peek (QUEUE *q);
                    /* dequeue without removing element*/
```

ADT



QUEUE using Linked List

```
struct qnode{
            int val;
            struct qnode *next;
          };

struct queue{
    struct qnode *qfront, *qrear;
     };

typedef struct queue QUEUE;
```

QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
void enqueue (QUEUE *q,int element)
{
  struct qnode *q1;
  q1=(struct qnode *)malloc(sizeof(struct
  qnode));
  q1->val= element;
  q1->next=q->qfront;
  q->qfront=q1;
}
```

QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
int size (queue *q)
{
  queue *q1;
  int count=0;
  q1=q;
  while(q1!=NULL) {
      q1=q1->next;
      count++;
  }
  return count;
}
```

QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
int peek (queue *q)
{
  queue *q1;
  q1=q;
  while(q1->next!=NULL)
      q1=q1->next;
  return (q1->val);
}
```

Implement this using QUEUE data structure.

QUEUE:: First-In-First-Out (FIFO)

Assume:: queue contains integer elements

```
int dequeue (queue *q)
{
   int val;
   queue *q1,*prev;
   q1=q;
   while(q1->next!=NULL) {
       prev=q1;
       q1=q1->next;
   }
   val=q1->val;
   prev->next=NULL;
   free(q1);
   return (val);
}
```

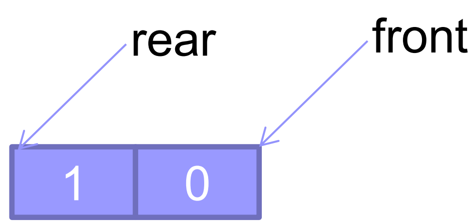
Implement this using QUEUE data structure.

Applications of Queues

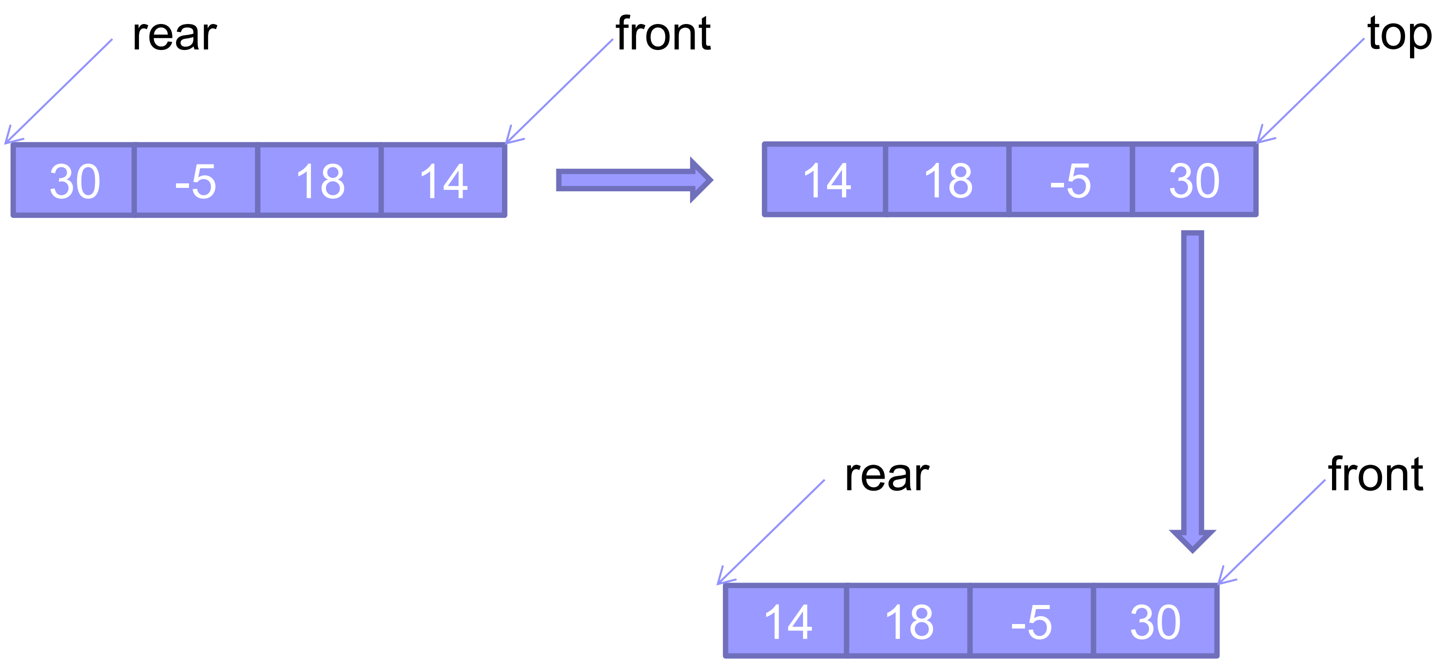
- **Direct applications**
  - Waiting lists.
  - Access to shared resources (e.g., printer).
  - Multiprogramming.
- **Indirect applications**
  - Auxiliary data structure for algorithms
  - Component of other data structures

Example 6: Print first *N* Fibonacci Numbers using a Queue

The queue initially contains 0 and 1

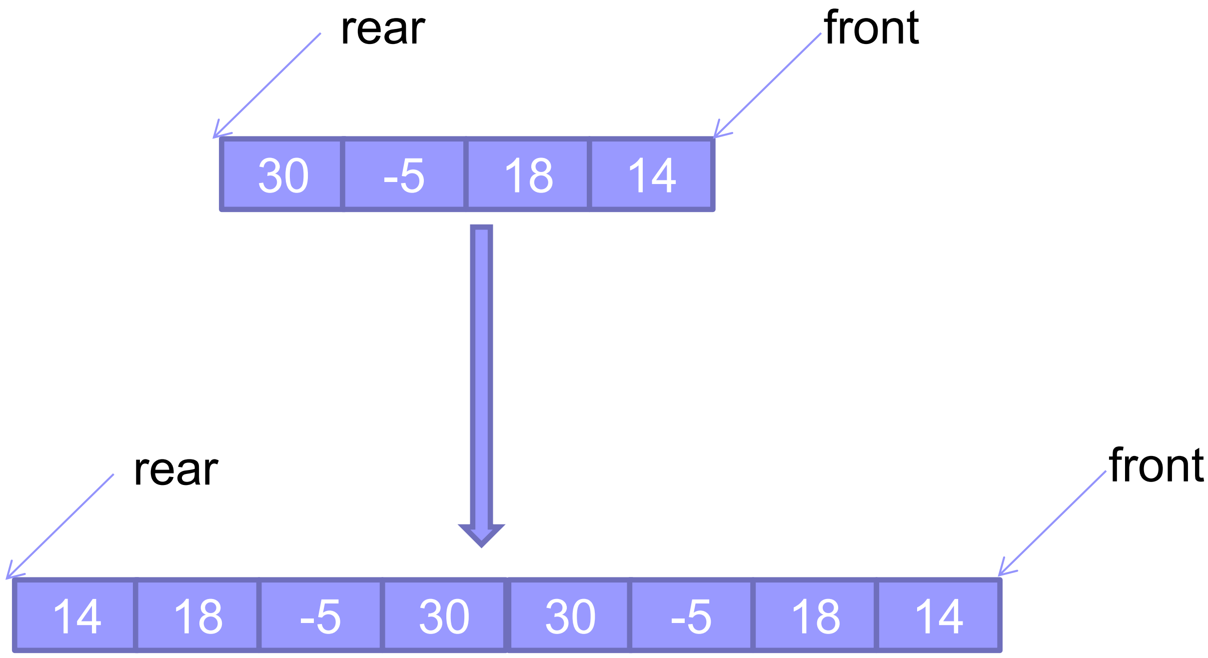


Example 7: Use a Stack to reverse a Queue



Example 8: Create a new Queue with given elements appended at the end of the Queue in a reverse order

* Hint- You can use a stack in order to achieve the outcome



Example 9: Implement a Stack using a Queue data structure

For a given stack create a same size array which you are going to use as a Queue.

Push and pop operation of stack's should be emulated with the Enqueue and Dequeue operation.

You can use an intermediate Queue for the above implementation.

Homework

- 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

Thank You!