QUEQUE

QUEUES

- Queue is a data structure that can be used to store data which can later be retrieved in the first in first out (FIFO) order.
- Queue is an ordered-list in which all the insertions and deletions are made at two different ends to maintain the FIFO order.
- The operations defined on a Queue are:
  1. Add - Store onto a Queue
  2. remove - retrieve (delete) from Queue
  3. Is_empty - check if the Queue is empty
  4. Is_Full - check if the Queue is full
- A Queue can be very easily implemented using arrays.
- Queue is implemented by maintaining one pointer to the *front* element in the Queue and another pointer pointing to the *rear* of the Queue.
- Insertions are made at the *rear* and deletions are made from the *front*.

QUEUES – ARRAY IMPLEMENTATION

```
class Queue {
public:
    Queue(int s = 10);                          // constructor - default size = 10
    ~Queue() {delete [ ] QueueArray; }          // destructor
    bool add (int);
    bool remove (int &);
    bool isFull()           {return MaxSize == size;}
    bool isEmpty()          {return size == 0; }
private:
    int MaxSize;                                // max Queue size
    int front, rear;
    int *QueueArray;
    int size;                                   // no. of elements in the Queue
};
```

```
Queue::Queue(int s)
{
    if (s <= 0) MaxSize = 10; else MaxSize = s;
    QueueArray = new int[MaxSize];
    size = 0;
    rear = -1;                  // points to the last element
    front = 0;                  // points to first element
}
```

```
bool Queue::add(int n)
{
    if (! isFull() ) {
        rear++;
        QueueArray[rear] = n;
        size++;
        return true;
    }
    else return false;
}
```

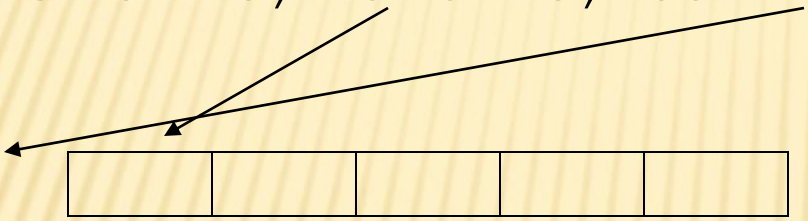
```
bool Queue::remove(int &n)
{
    if (! isEmpty() {
        n = QueueArray[front];
        front++;
        size--;
        return true;
    }
    else return false;
}
```

QUEUES

- Assume MaxSize = 5

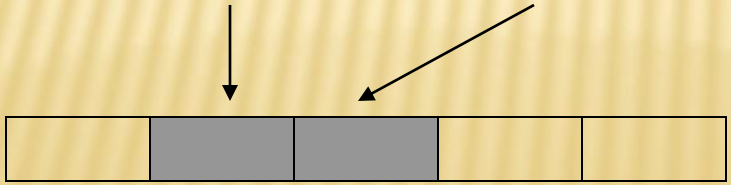
- Initial condition

  size = 0; front = 0; rear = -1;



- Add 3 elements, remove 1

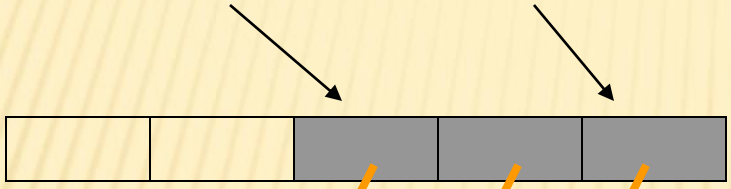
  size = 2; front = 1; rear = 2;



QUEUES

- Add 2 more, remove 1 more

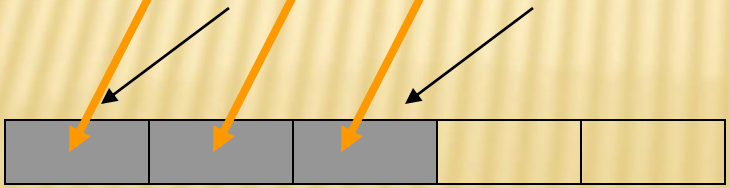
  size = 3; front = 2; rear = 4;



- Question: Is the Queue Full?
- Where to add the next element?

- Push everything back

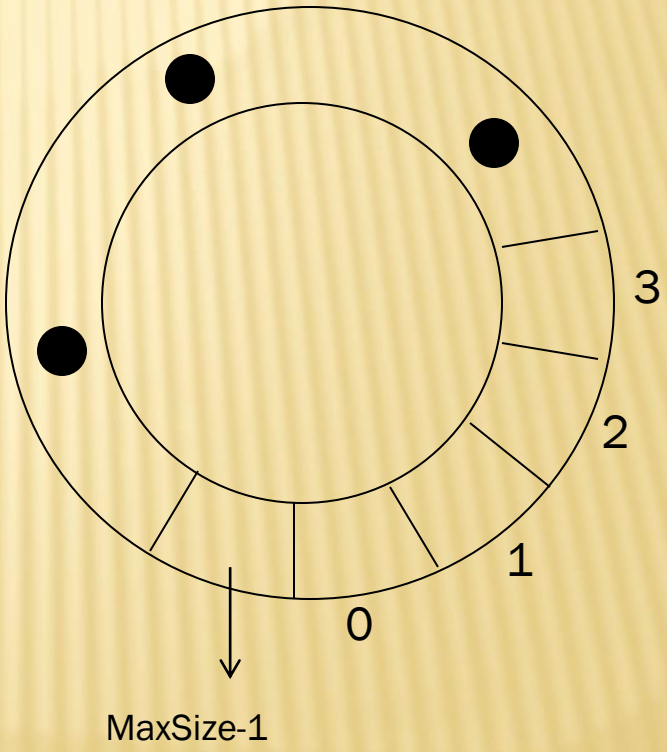
  size = 3; front = 0; rear = 2;



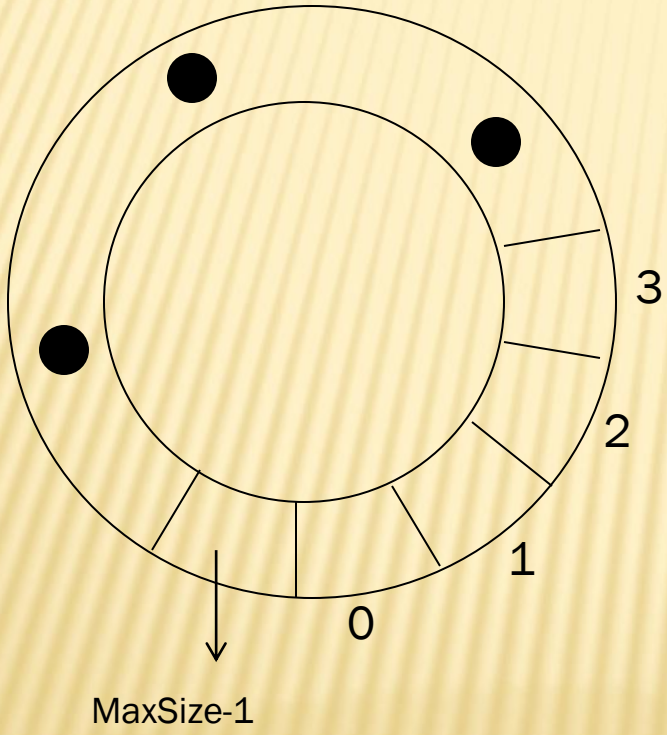
- Cost?
- O (size)

CIRCULAR IMPLEMENTATION

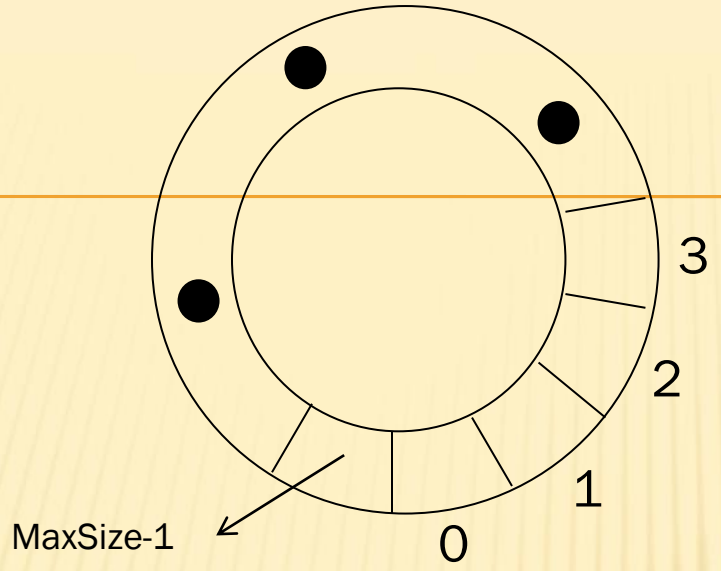
```
bool Queue::add(int n)
{
    if (! isFull() ) {
        rear++;
        if (rear == MaxSize)
            rear = 0;
        QueueArray[rear] = n;
        size++;
        return true;
    }
    else return false;
}
```



CIRCULAR IMPLEMENTATION



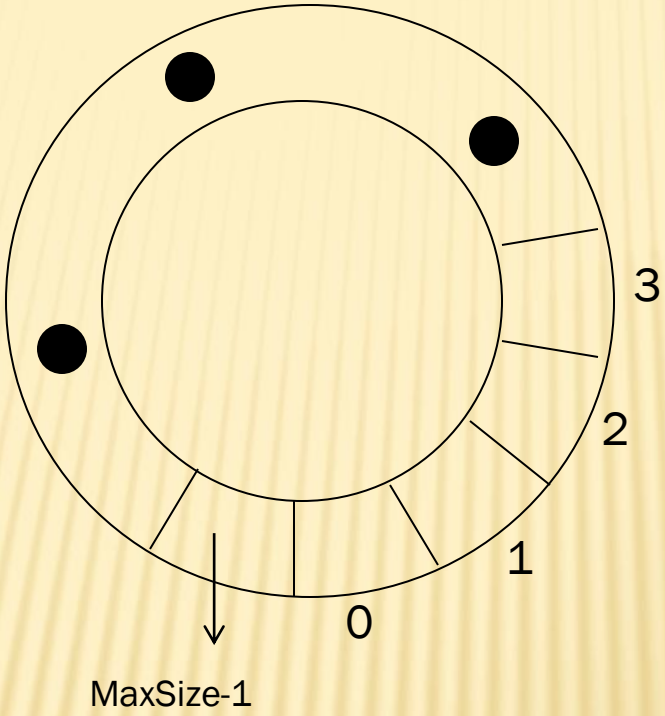

```
bool Queue::remove(int &n)
{
    if (! isEmpty() {
            n = QueueArray[front];
            front++;
            if (front == MaxSize)
                    front = 0;
            size--;
            return true;
    }
    else return false;
}
```




```
bool Queue::add(int n)
{
    if (! isFull() ) {
            rear++;
            if (rear == MaxSize)
                    rear = 0;
            QueueArray[rear] = n;
            size++;
            return true;
    }
    else return false;
}
```

```
bool Queue::remove(int &n)
{
    if (! isEmpty() {
            n = QueueArray[front];
            front++;
            if (front == MaxSize)
                    front = 0;
            size--;
            return true;
    }
    else return false;
}
```

CIRCULAR IMPLEMENTATION



Add → rear = (rear + 1) % MaxSize;

Remove → front = (front + 1) % MaxSize;