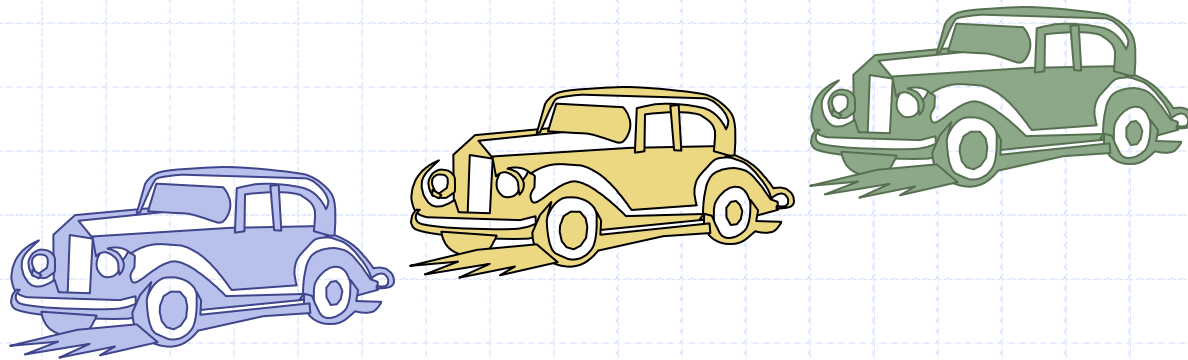


# Queues



# Outline and Reading

- ◆ The Queue ADT
- ◆ Implementation with a circular array
- ◆ Growable array-based queue
- ◆ Queue interface in C++

# The Queue ADT

- ◆ The **Queue** ADT stores arbitrary objects
- ◆ Insertions and deletions follow the first-in first-out scheme
- ◆ Insertions are at the rear of the queue and removals are at the front of the queue
- ◆ Main queue operations:
  - **enqueue**(Object o): inserts an element o at the end of the queue
  - **dequeue**(): removes and returns the element at the front of the queue
- ◆ Auxiliary queue operations:
  - **front**(): returns the element at the front without removing it
  - **size**(): returns the number of elements stored
  - **empty**(): returns a Boolean indicating whether no elements are stored
- ◆ Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an **EmptyQueueException**

# Applications of Queues

## ◆ Direct applications

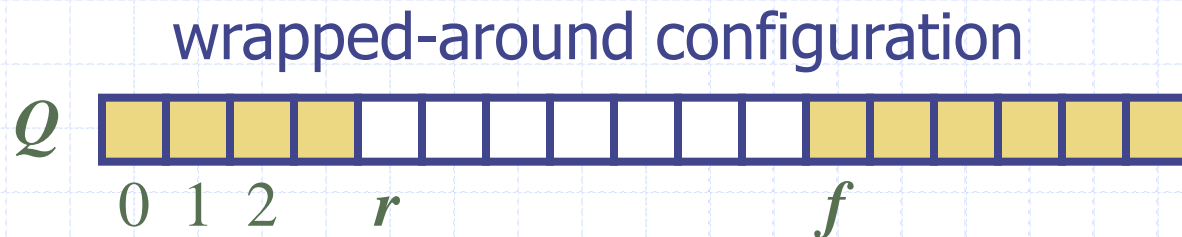
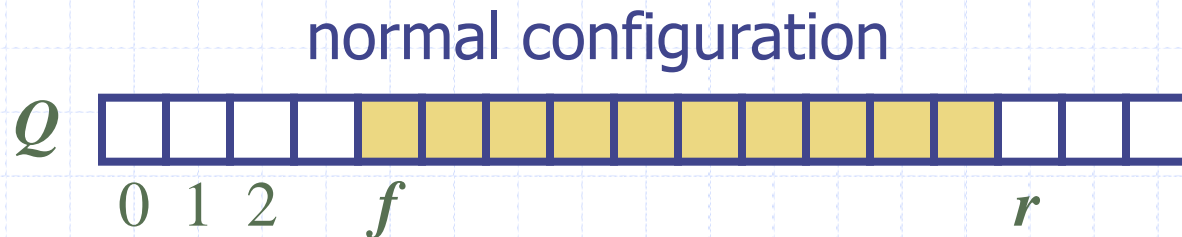
- Waiting lists, bureaucracy
- Access to shared resources (e.g., printer)
- Multiprogramming

## ◆ Indirect applications

- Auxiliary data structure for algorithms
- Component of other data structures

# Array-based Queue

- ◆ Use an array of size  $N$  in a circular fashion
- ◆ Two variables keep track of the front and rear
  - $f$  index of the front element
  - $r$  index immediately past the rear element
- ◆ Array location  $r$  is kept empty

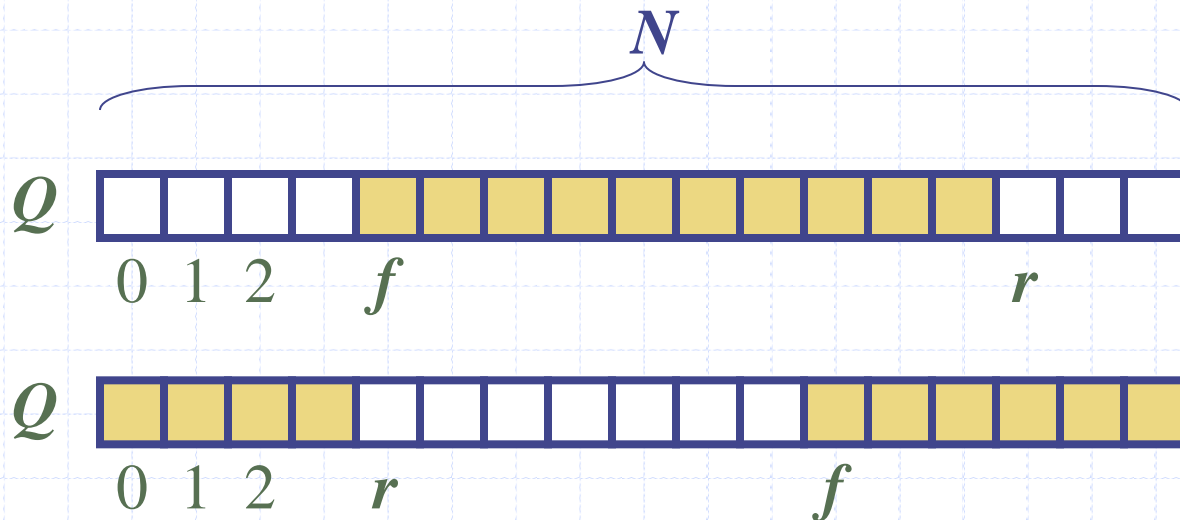


# Queue Operations

e.g.  $N=17, f=4, r=14$   
 $\text{size} = (17 - 4 + 14) \bmod 17$   
 $\text{size} = 27 \bmod 17$   
 $\text{size} = 10$

◆ Hint: use the modulo operator

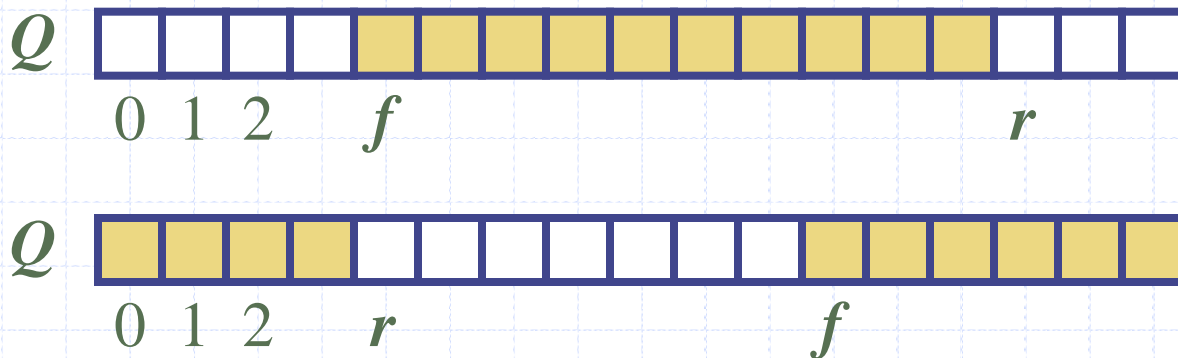
**Algorithm *size()***  
**return**  $(N - f + r) \bmod N$   
[or  $= ((N + r) - f) \bmod N$ ]  
**Algorithm *empty()***  
**return**  $(f = r)$



# Queue Operations (cont.)

- ◆ Operation enqueue throws an exception if the array is full
- ◆ This exception is implementation-dependent

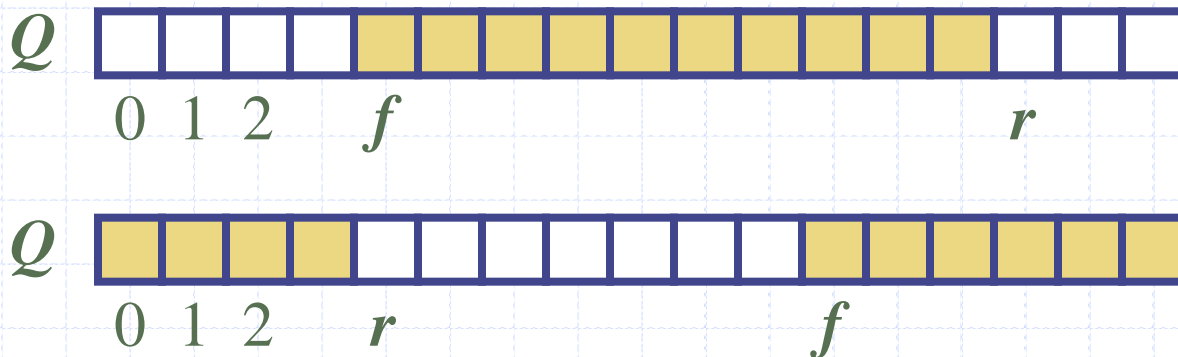
```
Algorithm enqueue(o)  
  if  $size() = N - 1$  then  
    throw FullQueueException  
  else  
     $Q[r] \leftarrow o$   
     $r \leftarrow (r + 1) \bmod N$ 
```



# Queue Operations (cont.)

- ◆ Operation `dequeue` throws an exception if the queue is empty
- ◆ This exception is specified in the queue ADT

```
Algorithm dequeue()  
  if isEmpty() then  
    throw EmptyQueueException  
  else  
     $o \leftarrow Q[f]$   
     $f \leftarrow (f + 1) \bmod N$   
  return  $o$ 
```





# Linked-List based Queue

◆ Will see later...

# Growable Array-based Queue

- ◆ In an enqueue operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one
- ◆ Similar to what we did for an array-based stack
- ◆ The enqueue operation has amortized running time
  - $O(n)$  with the incremental strategy
  - $O(1)$  with the doubling strategy

# Informal C++ Queue Interface

- ◆ Informal C++ interface for our Queue ADT
- ◆ Requires the definition of class `EmptyQueueException`
- ◆ A corresponding built-in STL class exists

```
template <typename Object>
class Queue {
public:
    int size();
    bool isEmpty();
    Object& front()
        throw(EmptyQueueException);
    void enqueue(Object o);
    Object dequeue()
        throw(EmptyQueueException);
};
```