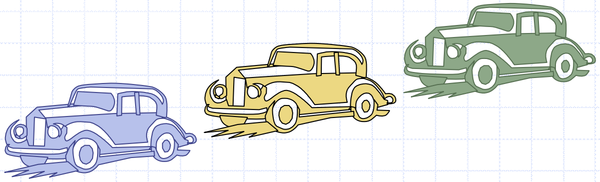
Queues



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): inserts an element at the end of the queue
  - dequeue(): removes the element at the front of the queue
- Auxiliary queue operations:
  - object front(): returns the element at the front without removing it
  - integer size(): returns the number of elements stored
  - boolean empty(): indicates whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an QueueEmpty

Example

Operation	***Output***	*Q*
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	–	(3)
enqueue(7)	–	(3, 7)
dequeue()	–	(7)
front()	*7*	(7)
dequeue()	–	()
dequeue()	*“error”*	()
empty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	–	(7, 3, 5)

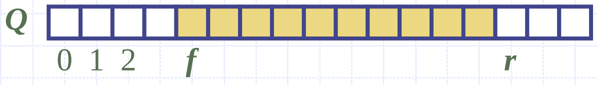
Applications of Queues

- 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

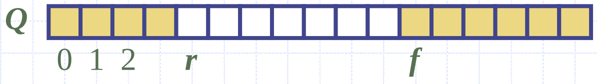
Array-based Queue

- Use an array of size N in a circular fashion
- Three variables keep track of the front and rear
  - f index of the front element
  - r index immediately past the rear element
  - n number of items in the queue

normal configuration



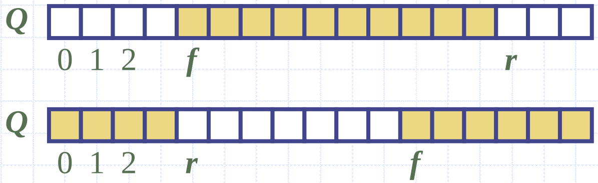
wrapped-around configuration



Queue Operations

- Use n to determine size and emptiness

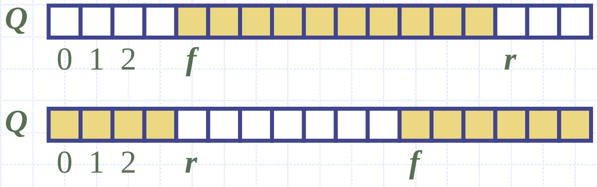
```
Algorithm size()
  return $n$

Algorithm empty()
  return $(n = 0)$
```



Queue Operations (cont.)

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

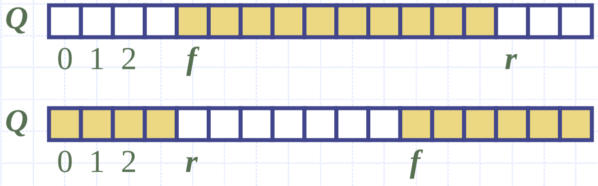
```
Algorithm enqueue(o)
  if $size() = N - 1$ then
    throw QueueFull
  else
    $Q[r] \leftarrow o$
    $r \leftarrow (r + 1) \bmod N$
    $n \leftarrow n + 1$
```



Queue Operations (cont.)

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

```
Algorithm dequeue()
  if empty() then
    throw QueueEmpty
  else
    $f \leftarrow (f + 1) \bmod N$
    $n \leftarrow n - 1$
```



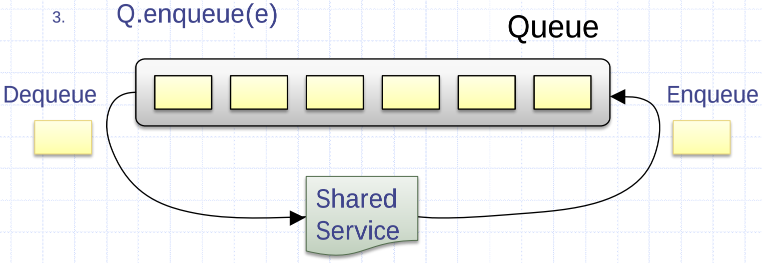
Queue Interface in C++

- C++ interface corresponding to our Queue ADT
- Requires the definition of exception QueueEmpty
- No corresponding built-in C++ class

```
template <typename E>
class Queue {
public:
   int size() const;
   bool empty() const;
   const E& front() const
      throw(QueueEmpty);
   void enqueue (const E& e);
   void dequeue()
      throw(QueueEmpty);
};
```

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  1. e = Q.front(); Q.dequeue()
  2. Service element e
  3. Q.enqueue(e)


Queue
Dequeue
Enqueue
Shared Service