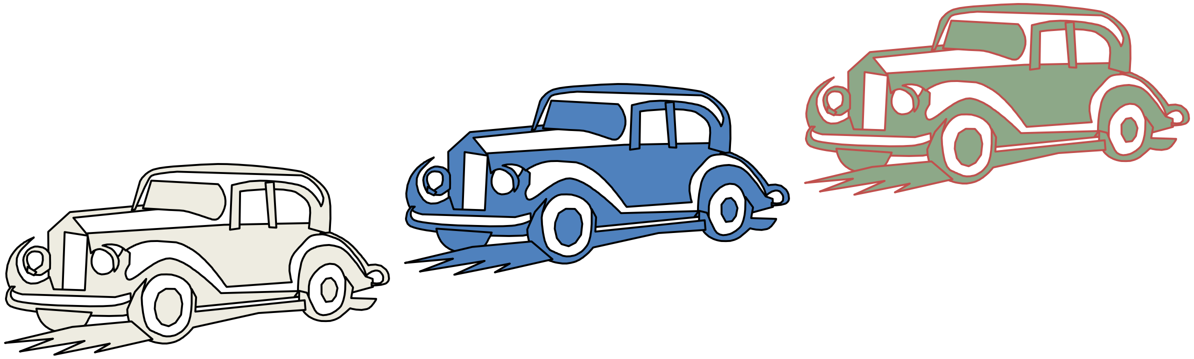
Queues
COL 106

Slides by Amit Kumar, Shweta Agrawal



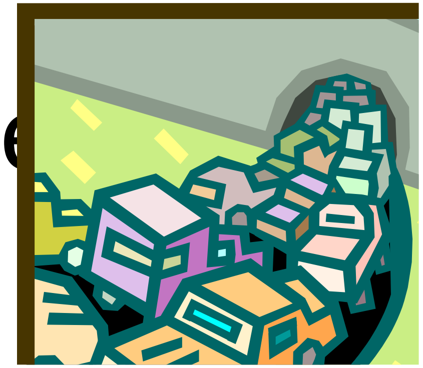
The Queue ADT

- The Queue ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out (FIFO) 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 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
  - isEmpty(): returns a Boolean value indicating whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyQueueException

Exercise: Queues

- Describe the output of the following series of queue operations
  - enqueue(8)
  - enqueue(3)
  - dequeue()
  - enqueue(2)
  - enqueue(5)
  - dequeue()
  - dequeue()
  - enqueue(9)
  - enqueue(1)

Applications of Queue



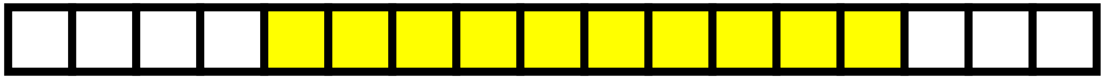
- Direct applications
  - Waiting lines
  - Access to shared resources (e.g., printer)

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

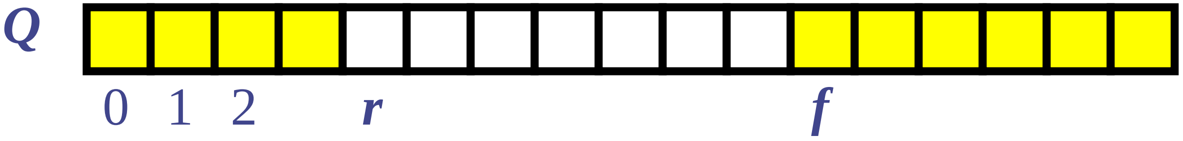
Array-based Queue

- 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

normal configuration



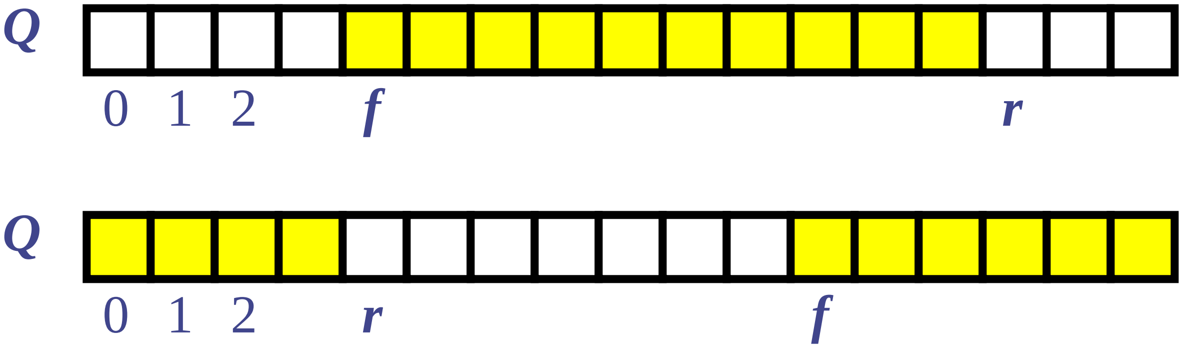
wrapped-around configuration



Queue Operations

- We use the modulo operator (remainder of division)

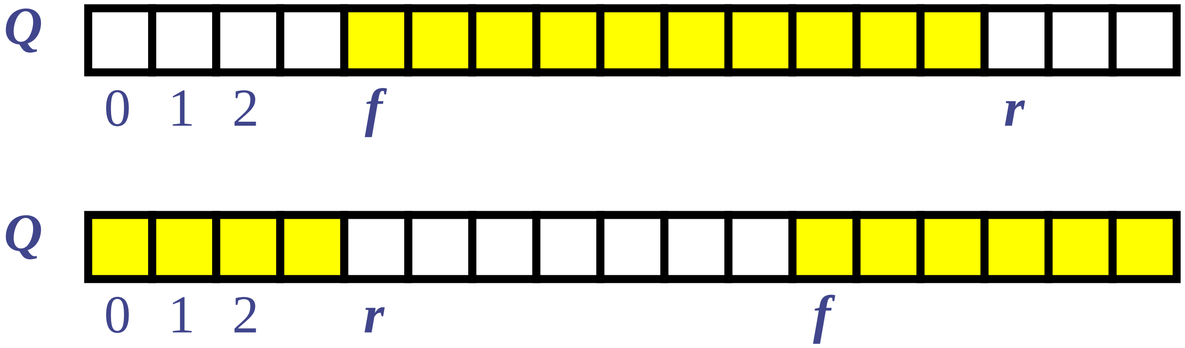
```
Algorithm size()
  return (N + r − f) mod N

Algorithm isEmpty()
  return (f = r)
```



Queue Operations (cont.)

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

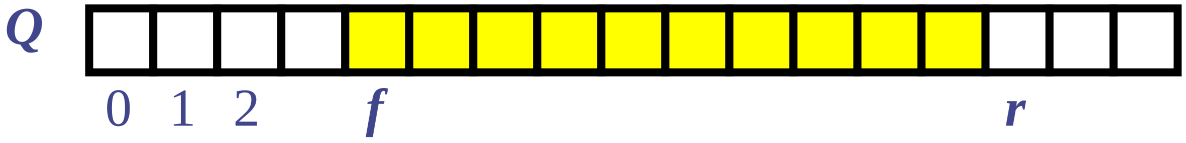
```
Algorithm enqueue(o)
  if size() = N - 1 then
    throw FullQueueException
   else
    Q[r] = o
    r = (r + 1) mod N
```

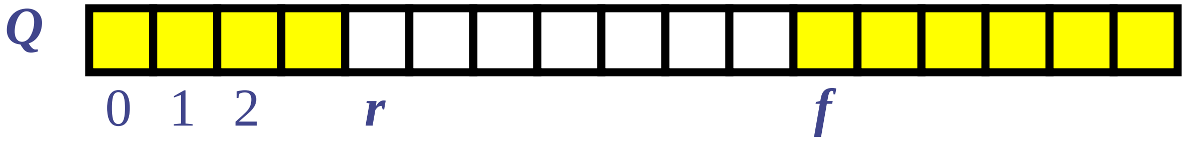


Queue Operations (cont.)

- 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 = Q[f]
    f = (f + 1) mod N
    return o
```





Performance and Limitations

- array-based implementation of queue ADT

- <u>Performance</u>
  - Let $\boldsymbol{n}$ be the number of elements in the queue
  - The space used is $\boldsymbol{O}(\boldsymbol{n})$
  - Each operation runs in time $\boldsymbol{O}(1)$
- <u>Limitations</u>
  - The maximum size of the queue must be defined *a priori* , and cannot be changed
  - Trying to enqueue an element into a full queue causes an implementation-specific exception

Growable Array-based Queue

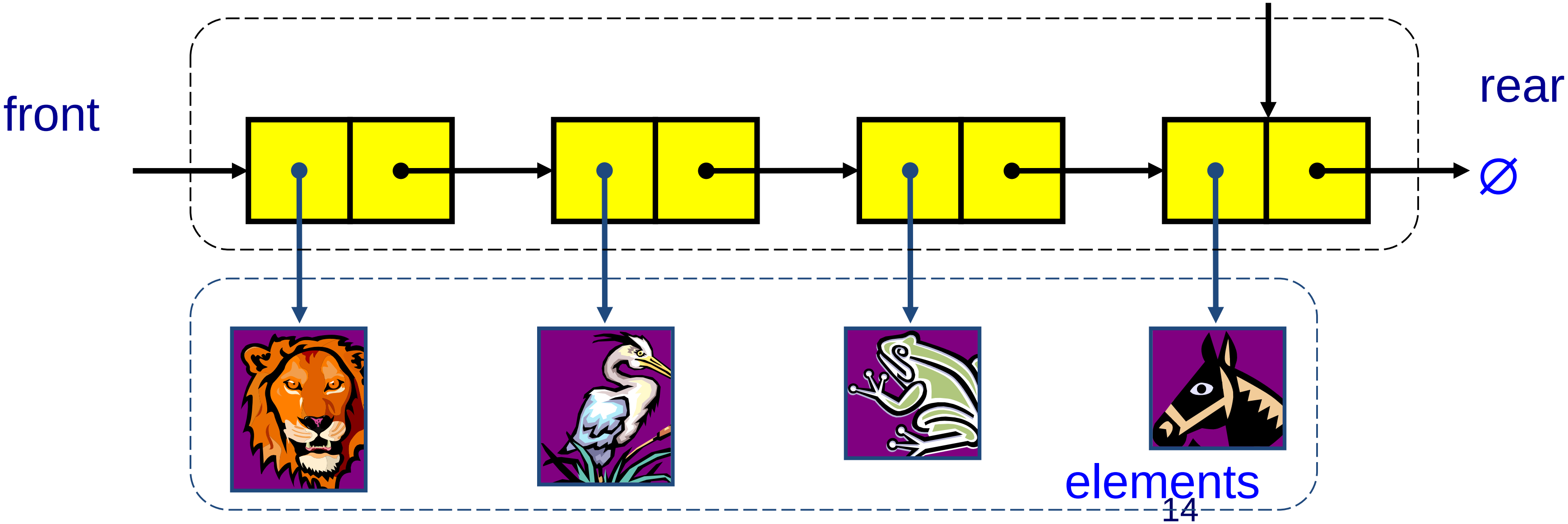
- In an enqueue operation, when the array is full, instead of throwing an exception, we can replace the array with a larger one
- Similar to what we did for an array-based stack
- The enqueue operation has amortized running time
  - $\boldsymbol{O}(\boldsymbol{n})$ with the incremental strategy
  - $\boldsymbol{O}(1)$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

Exercise

- Describe how to implement a queue using a singly-linked list
    - Queue operations: enqueue(x), dequeue(), size(), isEmpty()
    - For each operation, give the running time

Queue with a Singly Linked List

- We can implement a queue with a singly linked list
  - The front element is stored at the head of the list
  - The rear element is stored at the tail of the list
- The space used is $O(n)$ and each operation of the Queue ADT takes $O(1)$ time
- NOTE: we do not have the limitation of the array based implementation on the size of the stack b/c the size of the linked list is not fixed, I.e., the queue is NEVER full.



Queue Summary

- Queue Operation Complexity for Different

	Array Fixed-Size	Array Expandable (doubling strategy)	List Singly-Linked
dequeue()	O(1)	O(1)	O(1)
enqueue(o)	O(1)	O(n) Worst Case O(1) Best Case O(1) amortized analysis	O(1)
front()	O(1)	O(1)	O(1)
Size(), isEmpty()	O(1)	O(1)	O(1)

The Double-Ended Queue ADT (§5.3)

- The Double-Ended Queue, or Deque, ADT stores arbitrary objects. (Pronounced 'deck')
- Richer than stack or queue ADTs. Supports insertions and deletions at both the front and the end.
- Main deque operations:
  - insertFirst(object o): inserts element o at the beginning of the deque
  - insertLast(object o): inserts element o at the end of the deque
  - RemoveFirst(): removes and returns the element at the front of the queue
  - RemoveLast(): removes and returns the element at the end of the queue
- Auxiliary queue operations:
  - first(): returns the element at the front without removing it
  - last(): returns the element at the front without removing it
  - size(): returns the number of elements stored
  - isEmpty(): returns a Boolean value indicating whether no elements are stored
- Exceptions
  - Attempting the execution of dequeue or front on an empty queue throws an EmptyDequeException

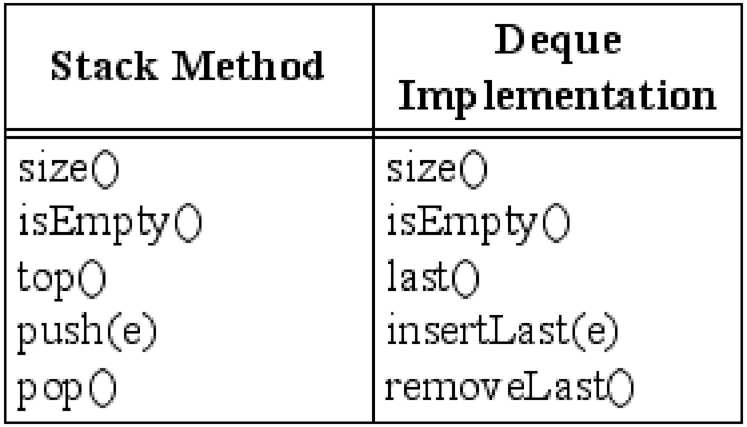
Deque Summary

- Deque Operation Complexity for Different

	Array Fixed-Size	Array Expandable (doubling strategy)	List Singly-Linked	List Doubly-Linked
removeFirst(), removeLast()	O(1)	O(1)	O(n) for one at list tail, O(1) for other	O(1)
insertFirst(o), InsertLast(o)	O(1)	O(n) Worst Case O(1) Best Case O(1) Average Case	O(1)	O(1)
first(), last	O(1)	O(1)	O(1)	O(1)
Size(), isEmpty()	O(1)	O(1)	O(1)	O(1)

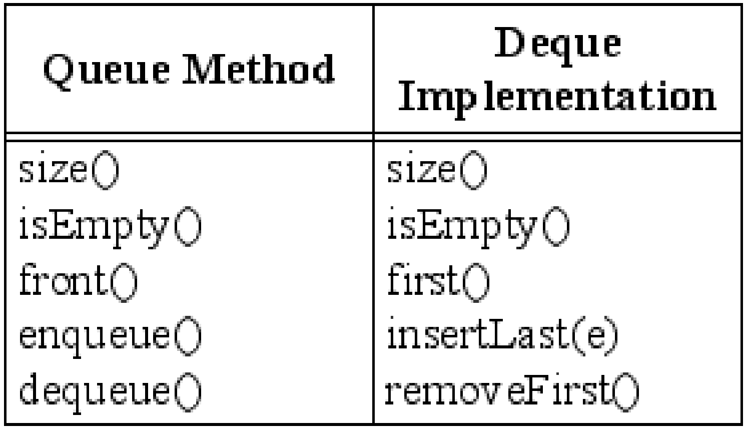
Implementing Stacks and Queues with Deques

Stacks with Deques:



Stack Method	Deque Implementation
size()	size()
isEmpty()	isEmpty()
top()	last()
push(e)	insertLast(e)
pop()	removeLast()

Queues with Deques:



Queue Method	Deque Implementation
size()	size()
isEmpty()	isEmpty()
front()	first()
enqueue()	insertLast(e)
dequeue()	removeFirst()

The Adaptor Pattern

- Using a deque to implement a stack or queue is an example of the adaptor pattern. Adaptor patterns implement a class by using methods of another class

- In general, adaptor classes specialize general classes

- Two such applications:

  -- Specialize a general class by changing some methods.

  Ex: implementing a stack with a deque.

  -- Specialize the types of objects used by a general class.

  Ex: Defining an IntegerArrayStack class that adapts ArrayStack to only store integers.