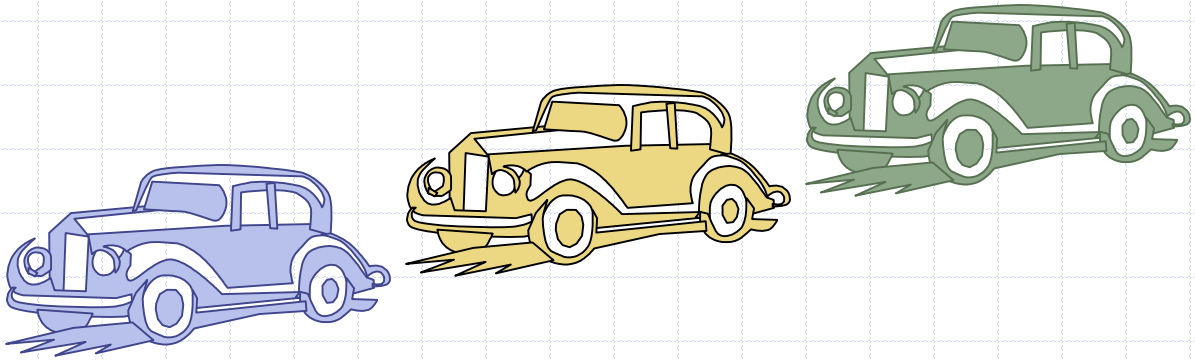
Queues



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

Example

Operation	Return Value	first ← Q ← last
Q.enqueue(5)	–	[5]
Q.enqueue(3)	–	[5, 3]
len(Q)	2	[5, 3]
Q.dequeue( )	5	[3]
Q.is_empty( )	False	[3]
Q.dequeue( )	3	[ ]
Q.is_empty( )	True	[ ]
Q.dequeue( )	"error"	[ ]
Q.enqueue(7)	–	[7]
Q.enqueue(9)	–	[7, 9]
Q.first( )	7	[7, 9]
Q.enqueue(4)	–	[7, 9, 4]
len(Q)	3	[7, 9, 4]
Q.dequeue( )	7	[9, 4]

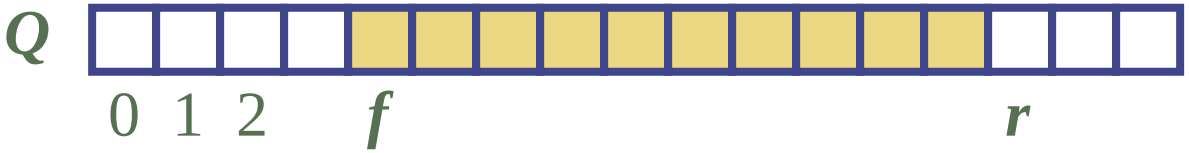
Applications of Queues

- 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

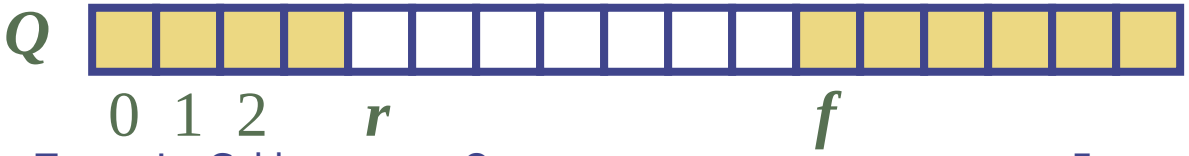
Array-based Queue

- 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

normal configuration



wrapped-around configuration

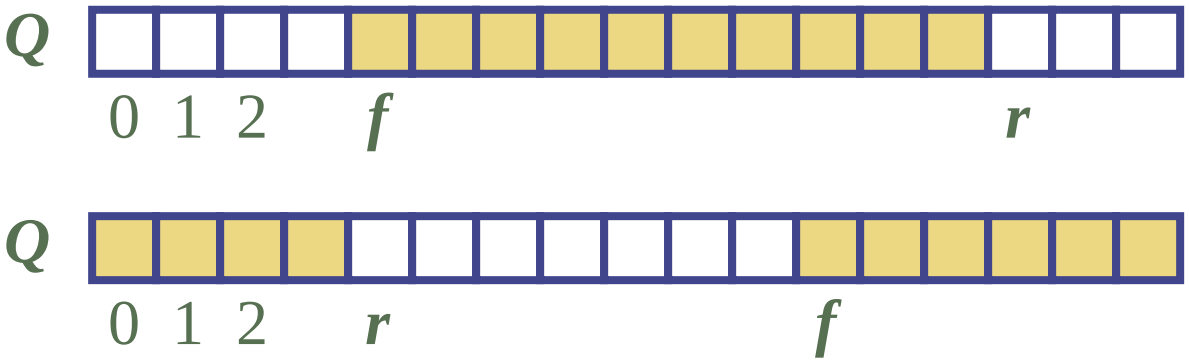


Queue Operations

- We use the modulo operator (remainder of division)

Algorithm ***size***()
  return $(N - f + r) \bmod N$

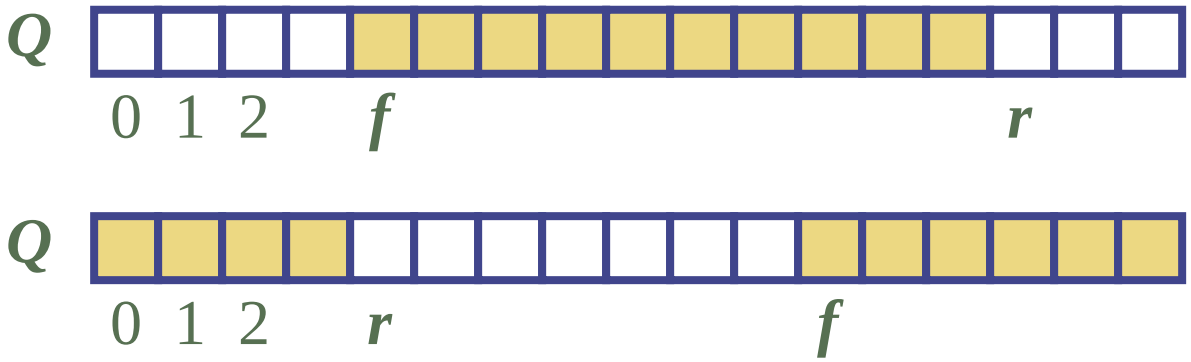
Algorithm ***isEmpty***()
  return $(f = r)$



Queue Operations (cont.)

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

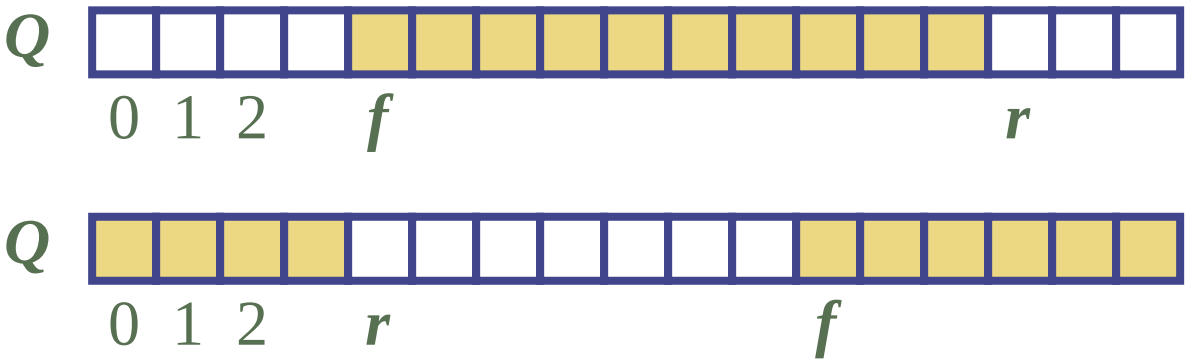
Algorithm *enqueue*(o)
  if *size*() = $N - 1$ **then**
    throw *FullQueueException*
  else
    $Q[r] \leftarrow o$
    $r \leftarrow (r + 1) \bmod N$



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



Queue in Python

- Use the following three instance variables:
  - _data: is a reference to a list instance with a fixed capacity.
  - _size: is an integer representing the current number of elements stored in the queue (as opposed to the length of the data list).
  - _front: is an integer that represents the index within data of the first element of the queue (assuming the queue is not empty).

Queue in Python (1/3)

```
class ArrayQueue:
  """FIFO queue implementation using a Python list as underlying storage."""
  DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

  def __init__(self):
    """Create an empty queue."""
    self._data = [None] * ArrayQueue.DEFAULT_CAPACITY
    self._size = 0
    self._front = 0

  def __len__(self):
    """Return the number of elements in the queue."""
    return self._size

  def is_empty(self):
    """Return True if the queue is empty."""
    return self._size == 0

  def first(self):
    """Return (but do not remove) the element at the front of the queue.

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[self._front]

  def dequeue(self):
    """Remove and return the first element of the queue (i.e., FIFO).

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    answer = self._data[self._front]
    self._data[self._front] = None                    # help garbage collection
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return answer
```

Queue in Python (2/3)

```
def enqueue(self, e):
  """Add an element to the back of queue."""
  if self._size == len(self._data):
    self._resize(2 * len(self.data))          # double the array size
  avail = (self._front + self._size) % len(self._data)
  self._data[avail] = e
  self._size += 1

def _resize(self, cap):                       # we assume cap >= len(self)
  """Resize to a new list of capacity >= len(self)."""
  old = self._data                            # keep track of existing list
  self._data = [None] * cap                   # allocate list with new capacity
  walk = self._front
  for k in range(self._size):                 # only consider existing elements
    self._data[k] = old[walk]                 # intentionally shift indices
    walk = (1 + walk) % len(old)              # use old size as modulus
  self._front = 0                             # front has been realigned
```

Queue in Python (3/3)

array_queue.py - C:\course\cs202\2015\ref\Codes\ch06\array_queue.py ...

File Edit Format Run Options Window Help

```
  def __repr__(self):
    """Create string representation of the favorites list."""
    return ', '.join(str(i) for i in self._data)

if __name__ == '__main__':
  Q = ArrayQueue()
  Q.enqueue(1); Q.enqueue(2); Q.enqueue(3)
  print(Q)
```

Ln: 79 Col: 0

Python 3.4.3 Shell

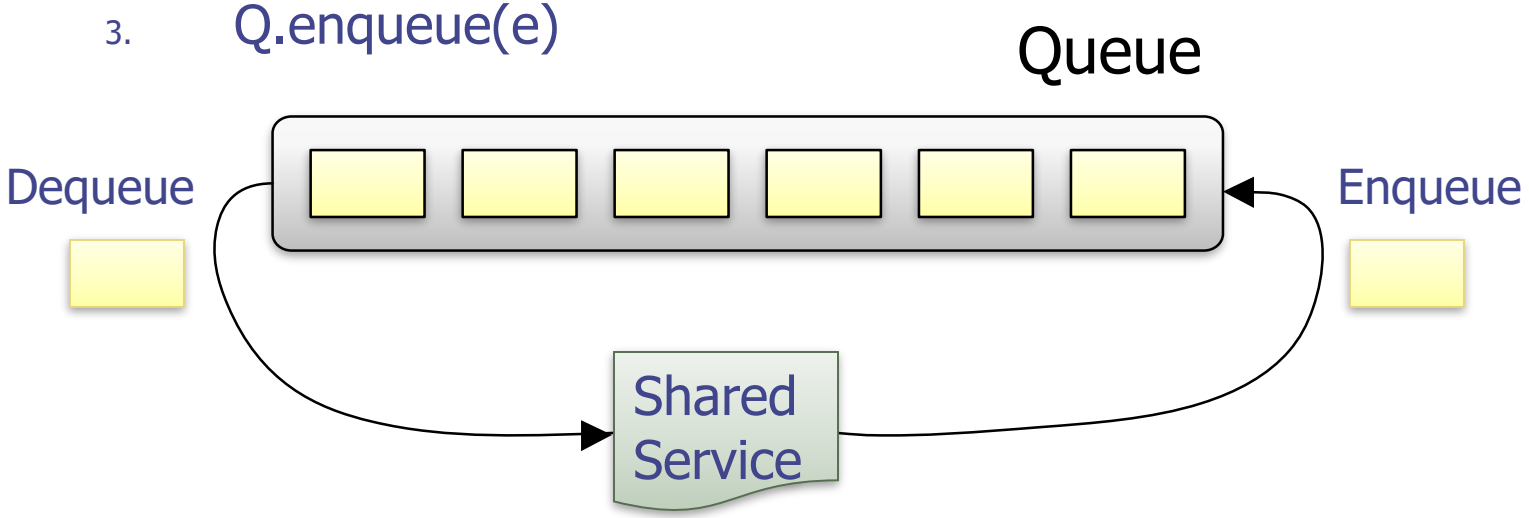
File Edit Shell Debug Options Window Help

```
>>>
1, 2, 3, None, None, None, None, None, None, None
>>>
```

Ln: 6 Col: 4

Application: Round Robin Schedulers

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



Double-Ended Queues

- queue-like data structure that supports insertion and deletion at both the front and the back of the queue.
- Such a structure is called a doubleended queue, or deque, which is usually pronounced "deck"
  - to avoid confusion with the dequeue method of the regular queue ADT, which is pronounced like the abbreviation "D.Q."
- An implementation of a deque class is available in Python's standard collections module.

Our Deque ADT	collections.deque	Description
len(D)	len(D)	number of elements
D.add_first( )	D.appendleft( )	add to beginning
D.add_last( )	D.append( )	add to end
D.delete_first( )	D.popleft( )	remove from beginning
D.delete_last( )	D.pop( )	remove from end
D.first( )	D[0]	access first element
D.last( )	D[−1]	access last element
	D[j]	access arbitrary entry by index
	D[j] = val	modify arbitrary entry by index
	D.clear( )	clea
	D.rotate(k)	circu
	D.remove(e)	rem
	D.count(e)	cou

Operation	Return Value	Deque
D.add_last(5)	–	[5]
D.add_first(3)	–	[3, 5]
D.add_first(7)	–	[7, 3, 5]
D.first( )	7	[7, 3, 5]
D.delete_last( )	5	[7, 3]
len(D)	2	[7, 3]
D.delete_last( )	3	[7]
D.delete_last( )	7	[ ]
D.add_first(6)	–	[6]
D.last( )	6	[6]
D.add_first(8)	–	[8, 6]
D.is_empty( )	False	[8, 6]
D.last( )	6	[8, 6]

Deque in Python (1/3)

array_deque.py - C:\course\cs202\2015\ref\Codes\ch06\array_deque.py (3.4.3)

File Edit Format Run Options Window Help

```python
from exceptions import Empty

class ArrayDeque:
  """FIFO queue implementation using a Python list as underlying storage."""
  DEFAULT_CAPACITY = 10          # moderate capacity for all new queues

  def __init__(self):
    """Create an empty queue."""
    self._data = [None] * ArrayDeque.DEFAULT_CAPACITY
    self._size = 0
    self._front = 0

  def __len__(self):
    """Return the number of elements in the queue."""
    return self._size

  def is_empty(self):
    """Return True if the queue is empty."""
    return self._size == 0

  def first(self):
    """Return (but do not remove) the element at the front of the queue.

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[self._front]

  #new added
  def last(self):
    if self.is_empty():
      raise Empty('Queue is empty')
    return self._data[(self._front+self._size-1 )% len(self._data)]
```

Ln: 37 Col: 0

Deque in Python (2/3)

array_deque.py - C:\course\cs202\2015\ref\Codes\ch06\array_deque.py (3.4.3)

File Edit Format Run Options Window Help

```
  def delete_last(self):
    if self.is_empty():
      raise Empty('Queue is empty')
    back = (self._front + self._size-1) % len(self._data)
    answer = self._data[back]
    self._data[back] = None          # help garbage collection
    self._size -= 1
    return answer

  def add_first(self, e):
    """Add an element to the front of queue."""
    if self._size == len(self._data):
      self._resize(2 * len(self.data))      # double the array size
    self._front = (self._front - 1) % len(self._data)
    self._data[self._front] = e
    self._size += 1

  #end added
  def delete_first(self):
    """Remove and return the first element of the queue (i.e., FIFO).

    Raise Empty exception if the queue is empty.
    """
    if self.is_empty():
      raise Empty('Queue is empty')
    answer = self._data[self._front]
    self._data[self._front] = None          # help garbage collection
    self._front = (self._front + 1) % len(self._data)
    self._size -= 1
    return answer

  def add_last(self, e):
    """Add an element to the back of queue."""
    if self._size == len(self._data):
      self._resize(2 * len(self.data))      # double the array size
    avail = (self._front + self._size) % len(self._data)
    self._data[avail] = e
    self._size += 1
```

Ln: 26 Col: 23

Deque in Python (3/3)

array_deque.py - C:\course\cs202\2015\ref\Codes\ch06\array_deque.py (3.4.3)

File Edit Format Run Options Window Help

```
  def _resize(self, cap):                    # we assume cap >= len(self)
    """Resize to a new list of capacity >= len(self)."""
    old = self._data                         # keep track of existing list
    self._data = [None] * cap                # allocate list with new capacity
    walk = self._front
    for k in range(self._size):              # only consider existing elements
      self._data[k] = old[walk]              # intentionally shift indices
      walk = (1 + walk) % len(old)           # use old size as modulus
    self._front = 0                          # front has been realigned

  def __repr__(self):
    """Create string representation of the favorites list."""
    #return ', '.join(str(i) for i in self._data)
    i = 0
    l = []
    while i < self._size:
      e = self._data[(self._front+i) % len(self._data)]
      l.append(e)
      i += 1
    return ', '.join(str(i) for i in l)

if __name__ == '__main__':
  D = ArrayDeque()
  D.add_last(5)
  D.add_first(3)
  D.add_first(7)
  #print(D)
  print(D.first())
  print(D.delete_last())
  print(len(D))
  print(D.delete_last())
  print(D.delete_last())
  D.add_first(6)
  print(D.last())
  D.add_first(8)
  print(D.is_empty())
  print(D.last())
```

Ln: 55 Col: 44

Python 3.4.3 Shell

File Edit Shell Debug Options Window Help

```
>>>
7
5
2
3
7
6
False
6
>>>
```

Ln: 84 Col: 4

Summary: Stack, Queue, Deque