# CSI33 Data Structures

Department of Mathematics and Computer Science
Bronx Community College

October 2, 2017

# Outline

1 Chapter 5: Stacks and Queues
- Queues

# Outline

1 Chapter 5: Stacks and Queues
   - Queues

# A Queue ADT

## A Container Class for First-In-First-Out Access

A queue is a list-like container with access restricted to both ends of the list. Items are added at the (the back of the queue) and removed from the front. In real life: “waiting in line” or “taking a number”.

- The enqueue method puts an item at the back of the queue.
- The dequeue method returns the item at the front, and removes it from the queue. (precondition: queue is not empty—`size > 0`)
- The front method returns that item (precondition: queue is not empty—`size > 0`)
- The size method returns the number of items in the queue.

# Simple Queue Application: A Palindrome Recognizer

## A Palindrome Recognizer

```
def isPalindrome(phrase):
   forward = Queue()
   reverse = Stack()
   extractLetters(phrase, forward, reverse)
   return sameSequence(forward, reverse)
```

# Simple Queue Application: A Palindrome Recognizer

### Phase I: Extract Letters

```
def extractLetters(phrase, q, s):
    for ch in phrase:
        if ch.isalpha():
            ch = ch.lower()
            q.enqueue(ch)
            s.push(ch)
```

# Simple Queue Application: A Palindrome Recognizer

### Phase II: Same Sequence

```
def sameSequence(q, s):
   while q.size() > 0:
      ch1 = q.dequeue()
      ch2 = s.pop()
      if ch1 != ch2:
         return False
   return True
```

# Queue Implementions

## A Python List Is Not An Efficient Queue

Both ends of the list must be used. The first item is either

- the front, and dequeue is $\Theta(n)$–every item must be moved down to delete the first item, or
- the back, and enqueue is $\Theta(n)$–every item must be moved up to insert the first item.

# Queue Implementions

## Linked List

This is the most flexible representation of a Queue ADT. By using a reference to the first node (front) and last node (back), a singly-linked list can perform enqueue and dequeue in constant time ($\Theta(1)$).

However, the links take up extra memory space.

# Queue Implementions

### Circular Array

A circular array avoids both the space inefficiency of links and the time inefficiency of the Python list (array) representation by not moving items. Instead, the front and back of the queue move, using changing indexes as markers.

- To enqueue a new item into the queue, the index marking the back of the queue is increased by one. The item that was in the back stays in the same position, but the new item goes behind it, into the new "back" position.
- Similarly, when an item is dequeued, the front of the queue is moved to the next item behind it.
- The array is "circular" because when either marker goes past the end of the array, it is put back at index zero.

# Simulation of Retail Store with One Checkout Register

## A Typical Queuing Simulation

To measure efficiency of service delivery, one runs a program that simulates these events:

- Random arrival times (customers finish shopping)
- Waiting for service (grocery checkout register)
- Random time to be serviced (number of items)

# Simulation of Retail Store with One Checkout Register

```
def genTestData(filename, totalTicks, maxItems, arrivalInterval):
   outfile = open(filename, "w")
   # step through the ticks
   for t in range(1,totalTicks):
      if random() < 1./arrivalInterval:
         # a customer arrives this tick
         # with a random number of items
         items = randrange(1, maxItems+1)
         outfile.write("%d %d\n" % (t, items))
   outfile.close()
```

# Simulation of Retail Store with One Checkout Register

```
class Customer(object):
   def __init__(self, arrivalTime, itemCount):
      self.arrivalTime = int(arrivalTime)
      self.itemCount = int(itemCount)
   def __repr__(self):
      return ("Customer(arrivalTime=%d, itemCount=%d)" %
         (self.arrivalTime, self.itemCount))
```

# Simulation of Retail Store with One Checkout Register

```
def createArrivalQueue(fname):
   q = Queue()
   infile = open(fname)
   for line in infile:
      time, items = line.split()
      q.enqueue(Customer(time,items))
   infile.close()
   return q
```

# Simulation of Retail Store with One Checkout Register

```
class CheckerSim(object):
   def __init__(self, arrivalQueue, avgTime):
      self.time = 0
      self.arrivals = arrivalQueue
      self.line = Queue()
      self.serviceTime = 0
      self.totalWait = 0
      self.maxWait = 0
      self.customerCount = 0
      self.maxLength = 0
      self.ticksPerItem = avgTime
```

# Simulation of Retail Store with One Checkout Register

```
def run(self):
   while (self.arrivals.size() > 0 or
   self.line.size() > 0 or
   self.serviceTime > 0):
      self.clockTick()
def averageWait(self):
   float(self.totalWait) / self.customerCount
def maximumWait(self):
   return self.maxWait
def maximumLineLength(self):
   return self.maxLength
```

# Simulation of Retail Store with One Checkout Register

```
def clockTick(self):
    self.time += 1
    while (self.arrivals.size() > 0 and
    self.arrivals.front().arrivalTime == self.time):
        self.line.enqueue(self.arrivals.dequeue())
        self.customerCount += 1
    self.maxLength = max(self.maxLength, self.line.size())
    if self.serviceTime > 0:
        self.serviceTime -= 1
    elif self.line.size() > 0:
        customer = self.line.dequeue()
        self.serviceTime = customer.itemCount * self.ticksPerItem
        waitTime = self.time - customer.arrivalTime
        self.totalWait += waitTime
        self.maxWait = max(self.maxWait, waitTime)
```