1. The Node class (in node.py) is used to dynamically create storage for a new item added to the stack. The LinkedStack class (in linked_stack.py) uses this Node class. Conceptually, a LinkedStack object would look like:

```
      _top:    data  next
    --+--  --+----
    |    |    |    
    a    b    c
   _size: 3
```

```
class Node:
    def __init__(self, initdata):
        self.data = initdata
        self.next = None
    def getData(self):
        return self.data
    def getNext(self):
        return self.next
    def setData(self, newdata):
        self.data = newdata
    def setNext(self, newnext):
        self.next = newnext
```

a) Complete the push, pop, and __str__ methods.

b) Stack methods big-oh’s?
   (Assume “n” items in stack)

   • constructor __init__:
   • push(item):
   • pop()
   • peek()
   • size()
   • isEmpty()
   • str()
A FIFO queue is basically what we think of as a waiting line.

The operations/methods on a queue object, say myQueue are:

<table>
<thead>
<tr>
<th>Method Call on myQueue object</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myQueue.dequeue()</td>
<td>Removes and returns the front item in the queue.</td>
</tr>
<tr>
<td>myQueue.enqueue(myItem)</td>
<td>Adds myItem at the rear of the queue</td>
</tr>
<tr>
<td>myQueue.peek()</td>
<td>Returns the front item in the queue without removing it.</td>
</tr>
<tr>
<td>myQueue.isEmpty()</td>
<td>Returns True if the queue is empty, or False otherwise.</td>
</tr>
<tr>
<td>myQueue.size()</td>
<td>Returns the number of items currently in the queue</td>
</tr>
<tr>
<td>str(myQueue)</td>
<td>Returns the string representation of the queue</td>
</tr>
</tbody>
</table>

2. Complete the following table by indicating which of the queue operations should have preconditions. Write “none” if a precondition is not needed.

<table>
<thead>
<tr>
<th>Method Call on myQueue object</th>
<th>Precondition(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>myQueue.dequeue()</td>
<td></td>
</tr>
<tr>
<td>myQueue.enqueue(myItem)</td>
<td></td>
</tr>
<tr>
<td>myQueue.peek()</td>
<td></td>
</tr>
<tr>
<td>myQueue.isEmpty()</td>
<td></td>
</tr>
<tr>
<td>myQueue.size()</td>
<td></td>
</tr>
<tr>
<td>str(myQueue)</td>
<td></td>
</tr>
</tbody>
</table>

3. The textbook’s Queue implementation use a Python list:

```python
class Queue:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def enqueue(self, item):
        self.items.insert(0,item)
    def dequeue(self):
        return self.items.pop()
    def peek(self):
        return self.items[0]
    def size(self):
        return len(self.items)
    def __str__(self):
```

a) Complete the __peek, and __str__ methods

b) What are the Queue methods big-oh’s? (Assume “n” items in the queue)
   - constructor __init__:
   - isEmpty() 
   - enqueue(item)
   - dequeue()
   - peek()
   - size()
   - str()
3. An alternate queue implementation using a linked structure (LinkedQueue class) would look like:

![LinkedQueue diagram]

a) Draw on the picture and number the steps for the enqueue method of the “normal” case (non-empty queue).

b) Write the enqueue method code for the “normal” case:

c) Starting with the empty queue below, draw the resulting picture after your “normal” case code executes.

![Empty LinkedQueue Object]

d) Fix your “normal” case code to handle the “special case” of an empty queue.
e) Draw on the above picture and number the steps for the `dequeue` method of the “normal” case (non-empty queue)

f) Write the `dequeue` method code for the “normal” case:


g) What “special case(s)” does the `dequeue` method code need to handle?

h) Draw the picture for each special case and number the steps for the `dequeue` method in the “special” case(s)

i) Combine the “normal” and special case(s) code for a complete `dequeue` method.

j) Complete the big-oh notation for the `LinkedQueue` methods: ("n" is the # items)

<table>
<thead>
<tr>
<th></th>
<th><strong>init</strong></th>
<th>enqueue(item)</th>
<th>dequeue( )</th>
<th>peek( )</th>
<th>size( )</th>
<th>isEmpty( )</th>
<th><strong>str</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Big-oh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>