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:

```
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?
(Imagine a stack with n items in it)

- constructor __init__: O(1)
- push(item): O(1)
- pop(): O(1)
- peek(): O(1)
- size(): O(1)
- isEmpty(): O(1)
- str(): O(n)
Steps for implementing "Linked" method "push"

1. Draw "normal-case" picture
   stuff in it already
2. Update picture for method
3. Number (steps)
4. Write "normal-case" code
5. Consider special cases:
   - empty stack
   - draw picture
   - run normal-case code
   - adjust as needed

Diagram:
- 1: Temp
- 2: Stack
- 3: Push
- 4: Size
- 5: Top
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>queue is not empty</td>
</tr>
<tr>
<td>myQueue.enqueue(myItem)</td>
<td>none</td>
</tr>
<tr>
<td>myQueue.peek()</td>
<td>none</td>
</tr>
<tr>
<td>myQueue.isEmpty()</td>
<td>none</td>
</tr>
<tr>
<td>myQueue.size()</td>
<td>none</td>
</tr>
<tr>
<td>str(myQueue)</td>
<td>none</td>
</tr>
</tbody>
</table>

3. The textbook's Queue implementation uses 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(0)

    def peek(self):
        return self.items[-1]

    def size(self):
        return len(self.items)

    def __str__(self):
        resultStr = "(Front) ... (Rear)"
        for index in range(len(self.items) - 1, -1, -1):
            resultStr += str(self.items[index])
        return resultStr
```

a) Complete the `peek` and `__str__` methods

b) What are the Queue methods big-oh's? (Assume "n" items in the queue)

- constructor `__init__`:
  \(O(1)\)
- `isEmpty()`:
  \(O(1)\)
- `enqueue(item)`:
  \(O(n)\)
- `dequeue()`:
  \(O(1)\)
- `peek()`:
  \(O(1)\)
- `size()`:
  \(O(1)\)
0. Draw normal case picture

1. Update picture for Pop

2. Number steps in picture

3. Write normal case code

   \[
   \text{temp} = \text{self}._\text{top}
   \]

   \[
   \text{self}._\text{top} = \text{self}._\text{top}.\text{get\_Next}()
   \]

   \[
   \text{self}._\text{size} += 1
   \]

   \[
   \text{return temp, get\_Data()}
   \]

4. Special cases

   - precondition
     - empty — raise exception
     - pop stack with one item

   \[
   \text{self}._\text{top} = \text{next}._\text{top}
   \]

   \[
   \text{return temp}
   \]
3. An alternate queue implementation using a linked structure (\texttt{LinkedQueue} class) would look like:

a) Draw on the picture and number the steps for the \texttt{enqueue} method of the "normal" case (non-empty queue)

b) Write the \texttt{enqueue} method code for the "normal" case:

\begin{enumerate}
\item \texttt{temp = Node(item)}
\item \texttt{self._rear.setNext(temp)}
\item \texttt{self._rear = temp}
\item \texttt{self._size += 1}
\end{enumerate}

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

\begin{itemize}
\item empty \texttt{LinkedQueue Object}
\item \texttt{front: [ ]}
\item \texttt{size: 0}
\item \texttt{rear: [ ]}
\end{itemize}

d) Fix your "normal" case code to handle the "special case" of an empty queue.