Linked Queue Push Operation

Assume that we have the following lines of code:

```c
myqueue queue1;       // Line 1
queue1.push(5);       // Line 2
queue1.push(8);       // Line 3
queue1.push(3);       // Line 4
```

The following sequence of diagrams shows how the `myqueue` object and its associated dynamic storage changes as these lines are executed.

**Figure 1:** The new, empty `myqueue` object `queue1` created in Line 1 of the code above. The `q_front` and `q_back` pointers are `nullptr`, while `q_size` is 0.

![Figure 1](image)

**Figure 2a:** The call to `push()` in Line 2 causes a new list `Node` to be allocated using the temporary pointer `new_node`. The node's `value` field is initialized to the value passed to `push()`, while its `next` field is initialized to `nullptr`.

![Figure 2a](image)

**Figure 2b:** Since the queue is currently empty, the pointer `q_front` is set to point at `new_node`. Then `q_back` is set to point at `new_node` and the `q_size` is incremented to 1.

![Figure 2b](image)
Figure 2c: When the \texttt{push()} method ends, the local variable \texttt{new_node} ceases to exist.

![Diagram showing the state of the queue before and after the push operation.](image)

Figure 3a: The call to \texttt{push()} in Line 3 causes a new list \texttt{Node} to be allocated using the temporary pointer \texttt{new_node}. The node's \texttt{value} field is initialized to the value passed to \texttt{push()}, while its \texttt{next} field is initialized to \texttt{nullptr}.

![Diagram showing the state of the queue before and after the push operation.](image)

Figure 3b: Since the queue is not empty, the pointer \texttt{q_back->next} is set to point at \texttt{new_node}. Then \texttt{q_back} is set to point at \texttt{new_node} and the \texttt{q_size} is incremented to 2.

![Diagram showing the state of the queue before and after the push operation.](image)

Figure 3c: When the \texttt{push()} method ends, the local variable \texttt{new_node} ceases to exist.

![Diagram showing the state of the queue before and after the push operation.](image)
**Figure 4a:** The call to `push()` in Line 4 causes a new list `Node` to be allocated using the temporary pointer `new_node`. The node's `value` field is initialized to the value passed to `push()`, while its `next` field is initialized to `nullptr`.

![Diagram 4a](image)

**Figure 4b:** Since the queue is not empty, the pointer `q_back->next` is set to point at `new_node`. Then `q_back` is set to point at `new_node` and the `q_size` is incremented to 3.

![Diagram 4b](image)

**Figure 4c:** When the `push()` method ends, the local variable `new_node` ceases to exist.

![Diagram 4c](image)

### Linked Queue Pop Operation

Assume that we then add the following lines of code after the code listed above:

```cpp
queue1.pop();       // Line 5
queue1.pop();       // Line 6
queue1.pop();       // Line 7
```

The following sequence of diagrams shows how the `myqueue` object and its associated dynamic storage changes as these lines are executed.
**Figure 5a:** The call to `pop()` in Line 5 creates the temporary pointer `del_node` and sets it to the value of `q_front`.

```
<table>
<thead>
<tr>
<th>q_front</th>
<th>q_back</th>
<th>q_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤</td>
<td>✤</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
del_node
   ▶
   5  ➔  8  ➔  3  ➔  X
```

**Figure 5b:** The pointer `q_front` is set to `q_front->next`. It now points to the 2nd node in the list. `q_front` is not `nullptr`, so `q_back` is not changed.

```
<table>
<thead>
<tr>
<th>q_front</th>
<th>q_back</th>
<th>q_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤</td>
<td>✤</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
del_node
   ▶
   5  ➔  8  ➔  3  ➔  X
```

**Figure 5c:** The node pointed to by `del_node` is deleted and `q_size` is decremented to 2.

```
<table>
<thead>
<tr>
<th>q_front</th>
<th>q_back</th>
<th>q_size</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤</td>
<td>✤</td>
<td>2</td>
</tr>
</tbody>
</table>
```

```
del_node
```

```
5    X
   ▶
   8  ➔  3  ➔  X
```
Figure 5d: When the pop() method ends, the local variable del_node ceases to exist.

Figure 6a: The call to pop() in Line 6 creates the temporary pointer del_node and sets it to the value of q_front.

Figure 6b: The pointer q_front is set to q_front->next. It now points to the 2nd node in the list. q_front is not nullptr, so q_back is not changed.
Figure 6c: The node pointed to by `del_node` is deleted and `q_size` is decremented to 1.

![Diagram showing deletion of a node in a queue.]

Figure 6d: When the `pop()` method ends, the local variable `del_node` ceases to exist.

![Diagram showing removal of `del_node` after pop().]

Figure 7a: The call to `pop()` in Line 7 creates the temporary pointer `del_node` and sets it to the value of `q_front`.

![Diagram showing creation of `del_node` with the value of `q_front`.]

Figure 7b: The pointer `q_front` is set to `q_front->next`. It is now `nullptr`. Since `q_front` is now `nullptr`, `q_back` is also set to `nullptr`.

![Diagram showing setting of `q_front` and `q_back` to `nullptr` after pop().]
Figure 7c: The node pointed to by del_node is deleted and q_size is decremented to 0.

Figure 7d: When the pop() method ends, the local variable del_node ceases to exist. The queue is now empty.