Linked Implementation Of A Queue, Reading

We have two options for linked queues:

1. add to the head and remove from the tail, or
2. add to the tail and remove from the head.

Let's see if there is any advantage or disadvantage to either. First, unlike with the stack, we'll need to interact with the tail end of the list, either to remove from there or to add there. You probably did not face this issue in your Comsc-110 introduction to linked lists, but you should have seen it in Comsc-165. It was shown at that time that if we maintain an additional pointer -- a "tail pointer" -- in addition to the head, then we always know where the end of the list is. We'll at least need that in our private data members:

```
Node* lastNode; // private data member
```

...and it will have to be zeroed out in the main constructor and clear functions.

```
lastNode = 0; // whenever the list is empty
```

If we add nodes at the head, as with the stack, then tail "points" to the node to be removed with pop. In that case we'd want to set its preceding node's next pointer to zero, *and* we'd want to point the tail pointer to that node. But wait -- we have no idea what is that node's memory location! Only *its* preceding node knows where it is. Unless we want to use a doubly-linked list just to solve this problem, popping the tail node may not work so well.

So how about option 2? We already know how to remove from the head -- the stack already does that. Pushing at the tail only needs to know where the tail node is, so that its next pointer can be made to point to the newly added node. This seems like a **winning combination** -- a **linked queue** with a **tail** pointer, **pushing at the tail** and popping at the head.

Push

We still have to create and fill a new node, just like Stack::push does. But instead of making it the new head, it's now the new tail, and *its* next pointer should be null. The old tail, while it's still the tail, should set its currently null next to point to the new node. That's it:

```
template <typename V>
void Queue<V>::push(const V& value)
{
  Node* temp = new Node;
  temp->value = value;
  temp->next = 0;
  if (lastNode) lastNode->next = temp;
  else firstNode = temp;
  lastNode = temp;
  ++siz;
}
```

or

```
template <typename V>
void Queue<V>::push(const V& value)
{
  Node* temp = new Node{value}; // C++11
  if (lastNode) lastNode->next = temp;
  else firstNode = temp;
  lastNode = temp;
  ++siz;
}
```

C++11 allows curly-brace initialization with "new"

Note the if-else, because if there is no tail, then the list is empty and the new node is both the head *and* the tail.

Copy Constructor And Assignment Operator

The temporary tail pointer in stack versions of these functions is no longer temporary -- it's now a data member. So:

```
~~Node*~~ lastNode = 0; // ~~temporary~~ tail
```

Pop And Clear

The tail pointer needs to be zeroed out in Queue::clear, and before Queue::pop ends, it needs to check if the siz is now zero, because if it is, the tail pointer needs to be zeroed out.

Node, Peek, and Capacity

These are all unaffected by the addition of a tail pointer.