Queues and Priority Queue Implementations

Chapter 14

Content

- Implementations of the ADT Queue
- An Implementation of the ADT Priority Queue

An Implementation That Uses the ADT List

- View Listing 14-1 header for class **`ListQueue`**
- Note implementation, Listing 14-2

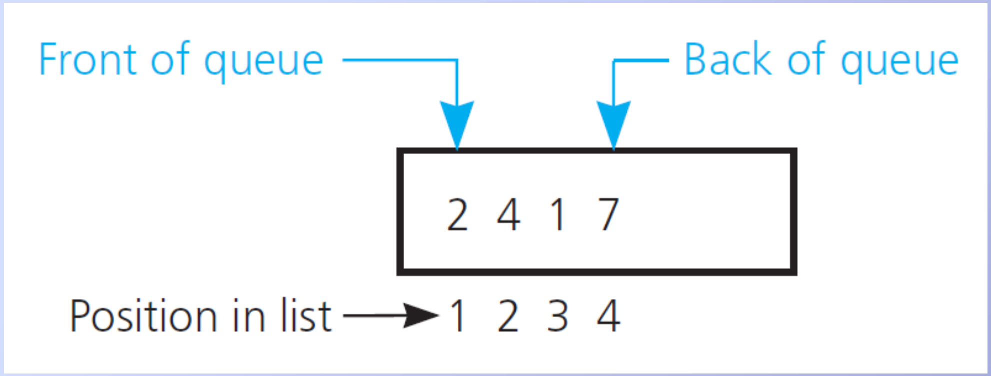


FIGURE 14-1 An implementation of the ADT queue that stores its entries in a list

A Link-Based Implementation

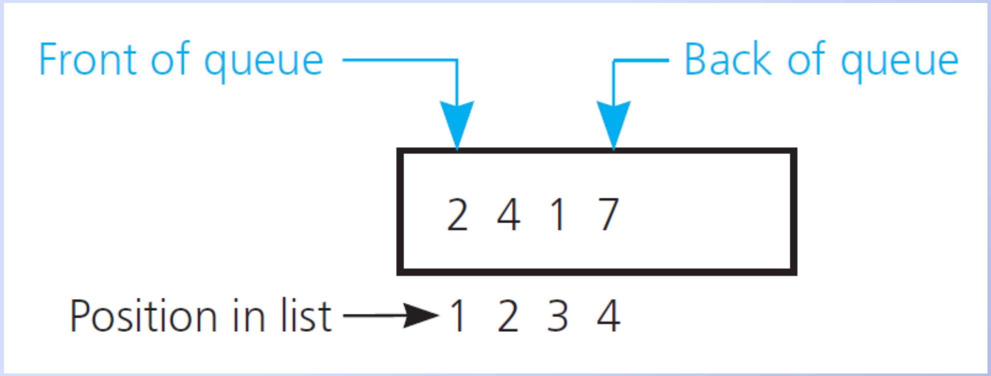


FIGURE 14-1 An implementation of the ADT queue that stores its entries in a list

A Link-Based Implementation

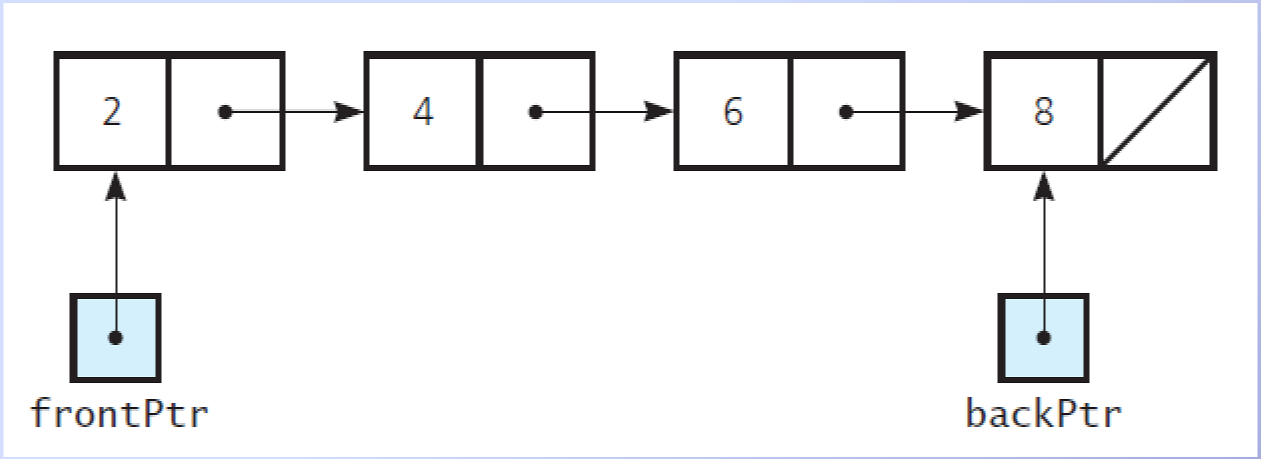
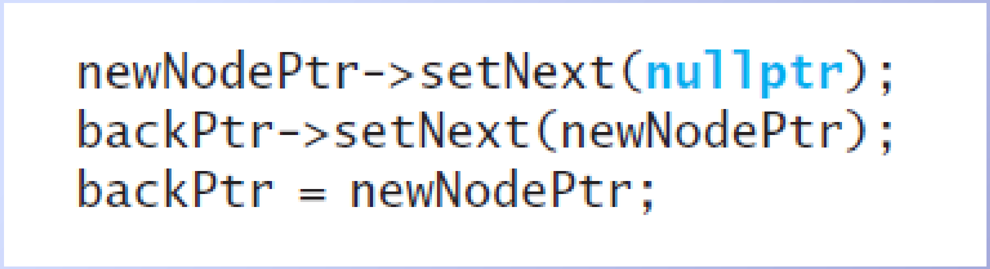


FIGURE 14-3 A circular chain of linked nodes with one external pointer

A Link-Based Implementation

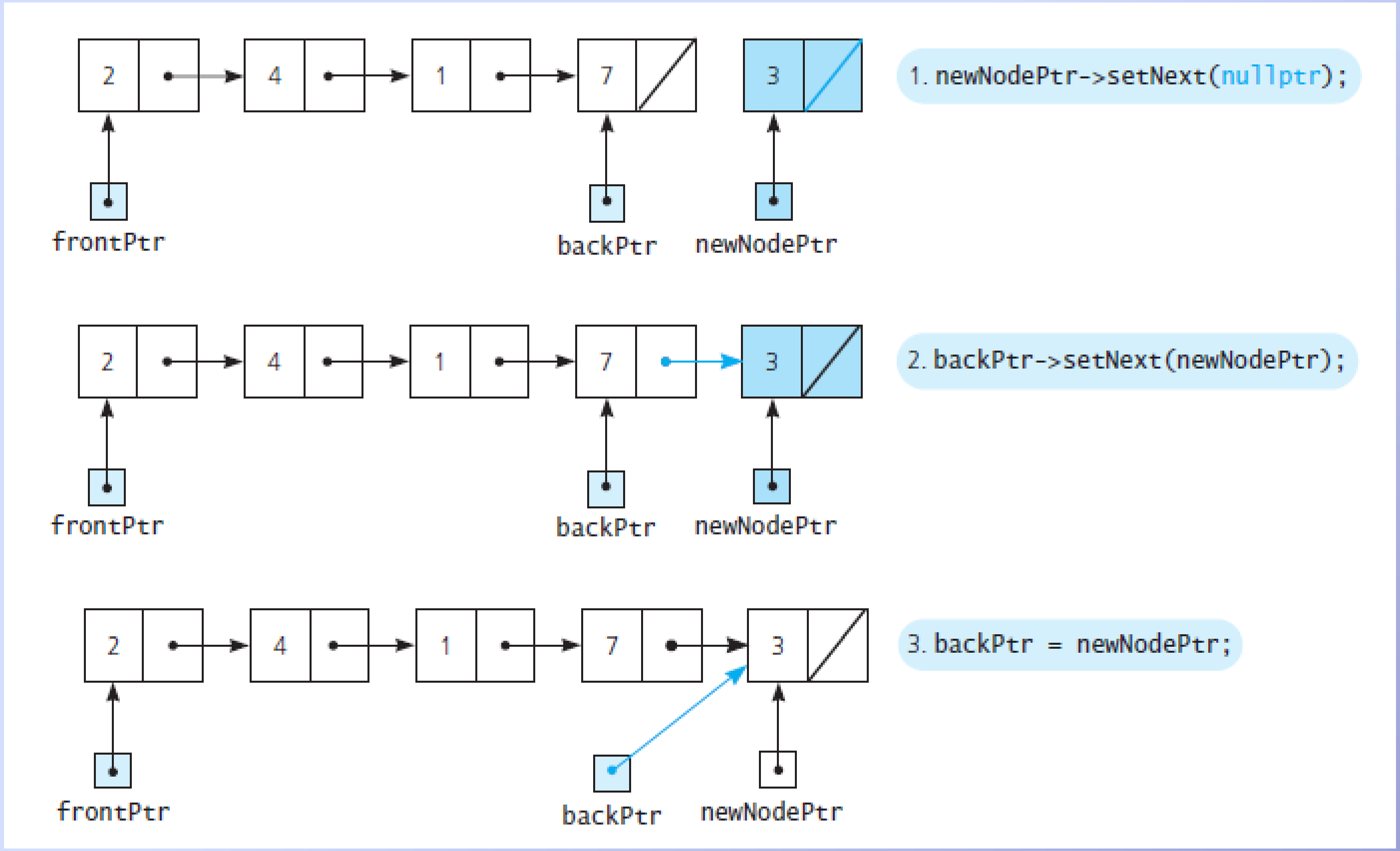
- View header file for class **`LinkedQueue`**, Listing 14-3

- The enqueue method to insert a new node



```
newNodePtr->setNext(nullptr);
backPtr->setNext(newNodePtr);
backPtr = newNodePtr;
```

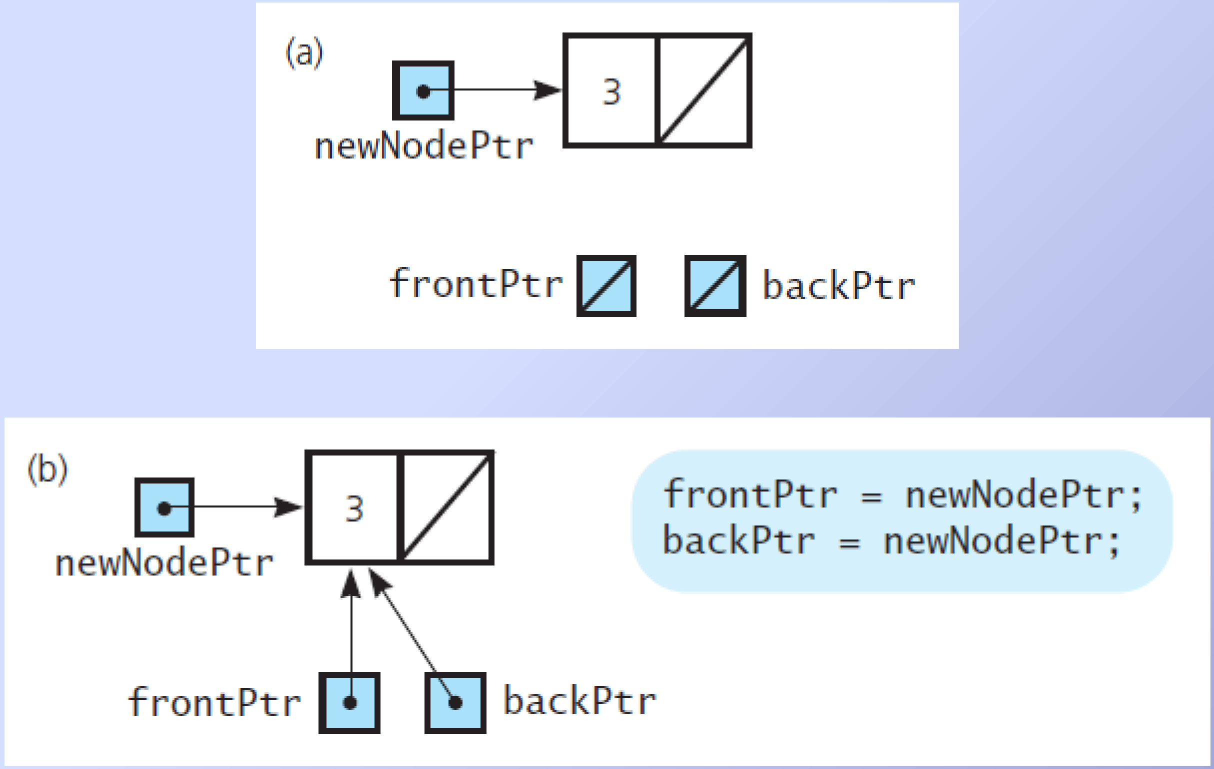
A Link-Based Implementation



```
1. newNodePtr->setNext(nullptr);
2. backPtr->setNext(newNodePtr);
3. backPtr = newNodePtr;
```

FIGURE 14-4 Adding an item to a nonempty queue

A Link-Based Implementation



```
frontPtr = newNodePtr;
backPtr = newNodePtr;
```

FIGURE 14-5 Adding an item to an empty queue: (a) before enqueue; (b) after enqueue

A Link-Based Implementation

- Definition for the method **`enqueue`**

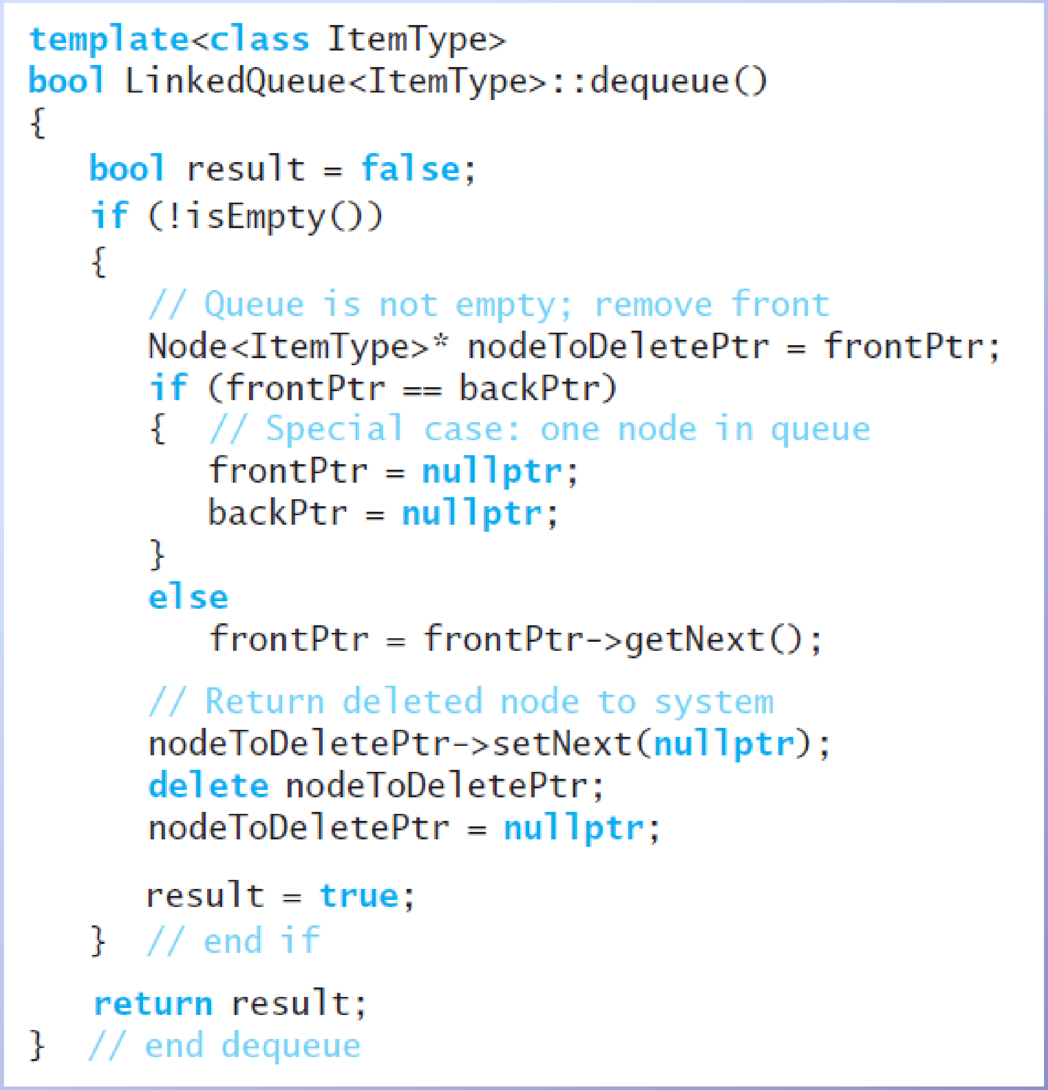
```cpp
template<class ItemType>
bool LinkedQueue<ItemType>::enqueue(const ItemType& newEntry)
{
   Node<ItemType>* newNodePtr = new Node<ItemType>(newEntry);

   // Insert the new node
   if (isEmpty())
      frontPtr = newNodePtr;           // The queue was empty
   else
      backPtr->setNext(newNodePtr);    // The queue was not empty

   backPtr = newNodePtr;               // New node is at back
   return true;
}  // end enqueue
```

A Link-Based Implementation

- Definition of the method **`dequeue`**



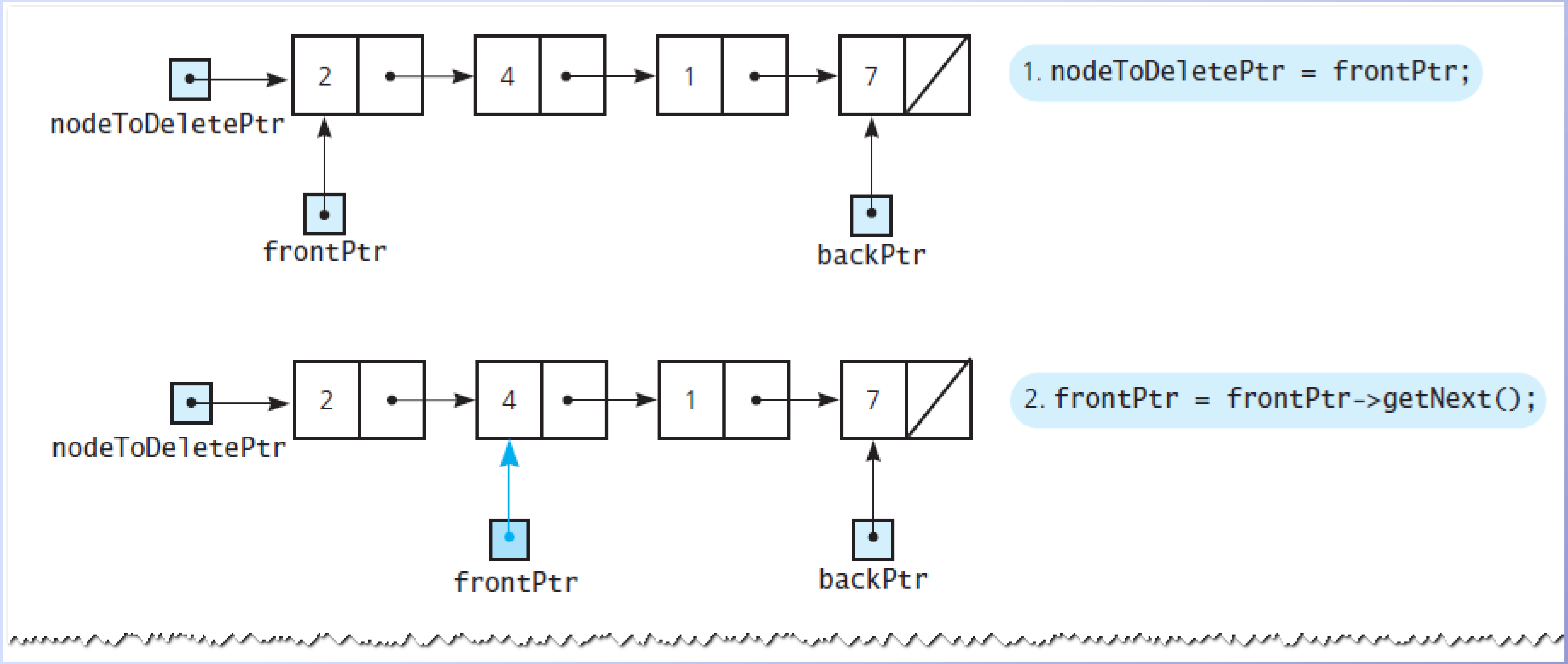
```
template<class ItemType>
bool LinkedQueue<ItemType>::dequeue()
{
   bool result = false;
   if (!isEmpty())
   {
      // Queue is not empty; remove front
      Node<ItemType>* nodeToDeletePtr = frontPtr;
      if (frontPtr == backPtr)
      {  // Special case: one node in queue
         frontPtr = nullptr;
         backPtr = nullptr;
      }
      else
         frontPtr = frontPtr->getNext();

      // Return deleted node to system
      nodeToDeletePtr->setNext(nullptr);
      delete nodeToDeletePtr;
      nodeToDeletePtr = nullptr;

      result = true;
   }  // end if

   return result;
}  // end dequeue
```

A Link-Based Implementation



1. `nodeToDeletePtr = frontPtr;`

2. `frontPtr = frontPtr->getNext();`

FIGURE 14-6 Removing an item from a queue of more than one item

A Link-Based Implementation

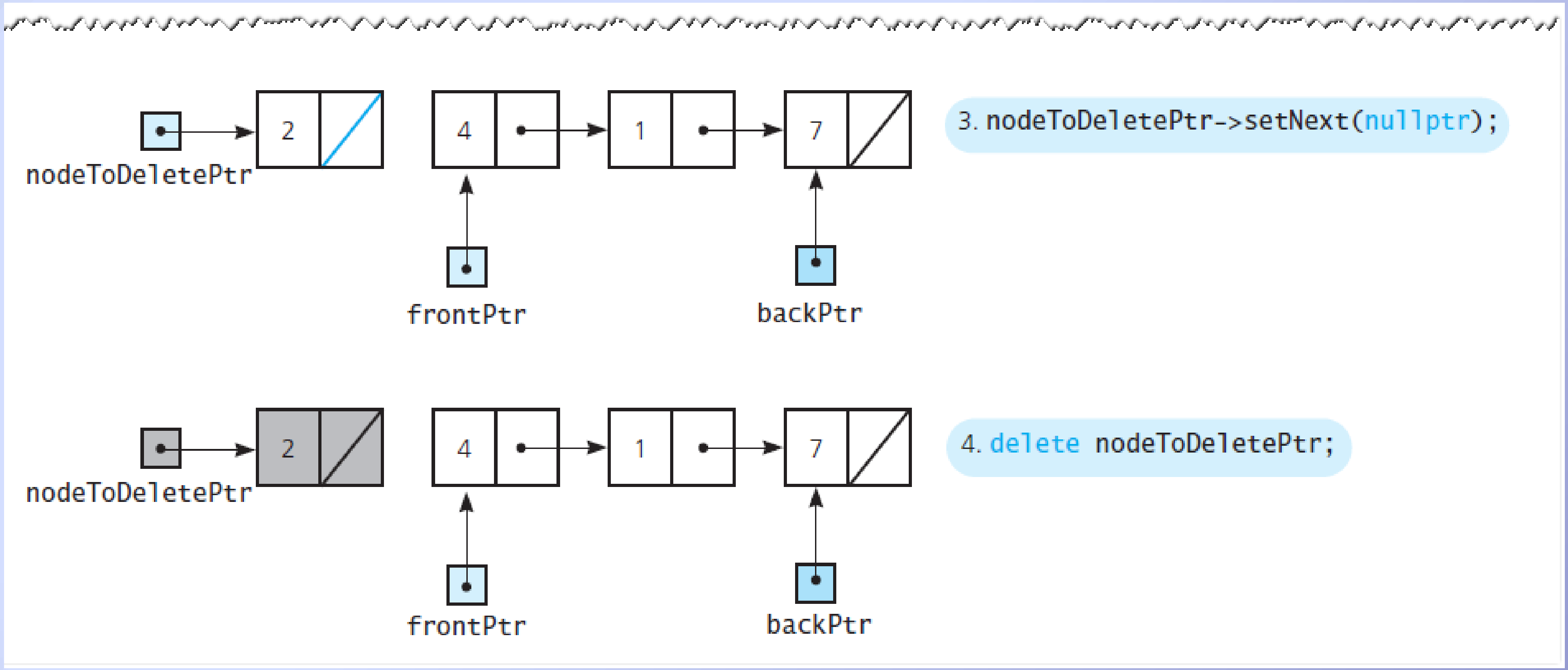
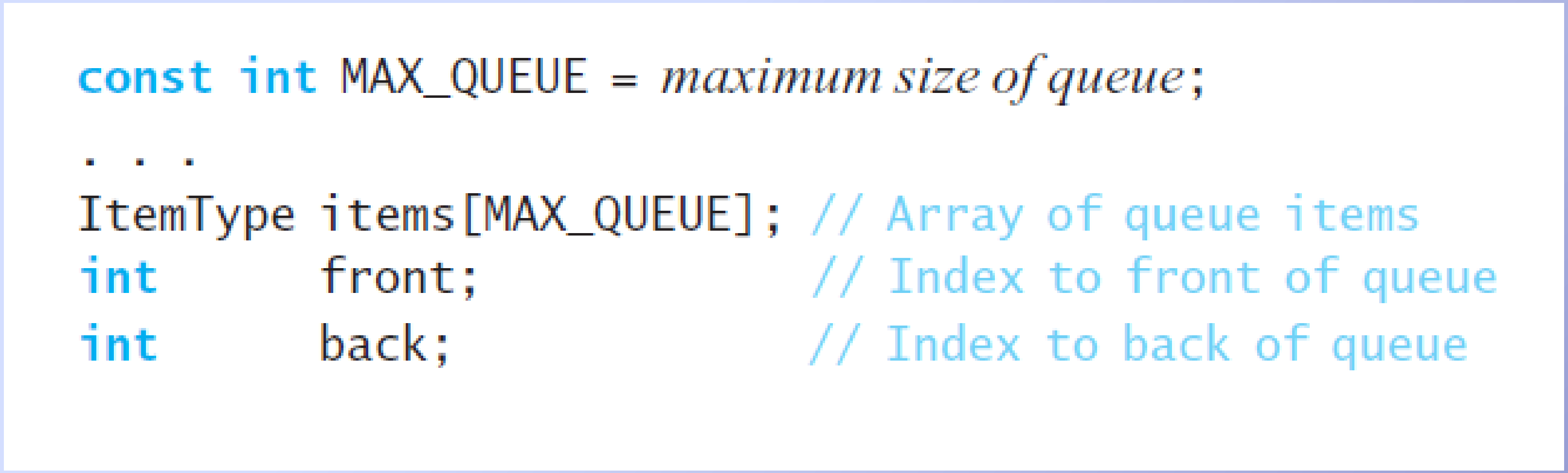


FIGURE 14-6 Removing an item from a queue of more than one item

An Array-Based Implementation

- Possible (naïve) definition



```
const int MAX_QUEUE = maximum size of queue;
. . .
ItemType items[MAX_QUEUE]; // Array of queue items
int      front;            // Index to front of queue
int      back;             // Index to back of queue
```

An Array-Based Implementation

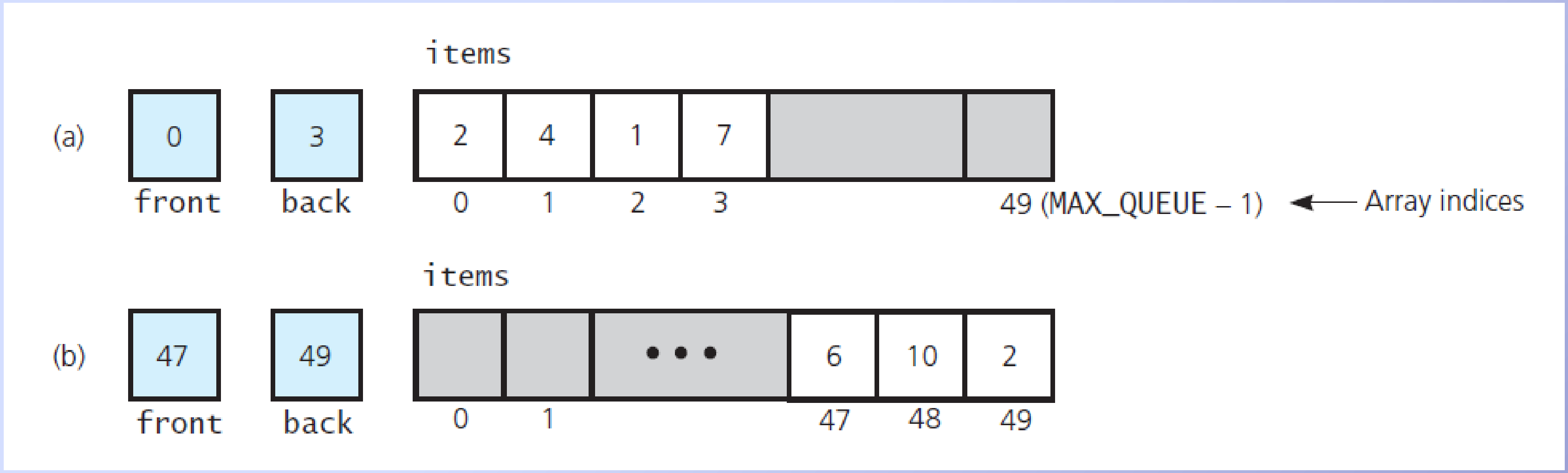


FIGURE 14-7 (a) A naive array-based implementation of a queue; (b) rightward drift can cause the queue to appear full

An Array-Based Implementation

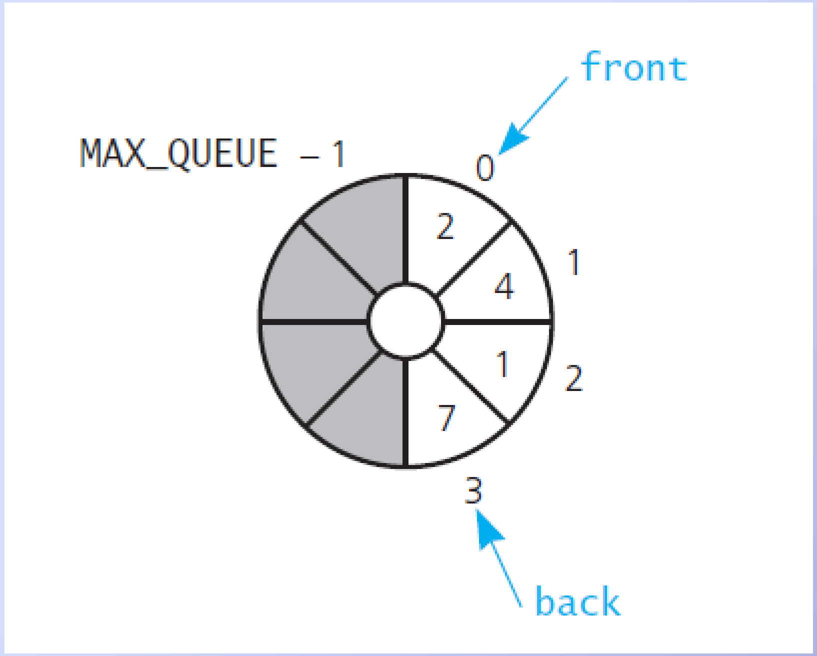


FIGURE 14-8 A circular array as an implementation of a queue

An Array-Based Implementation

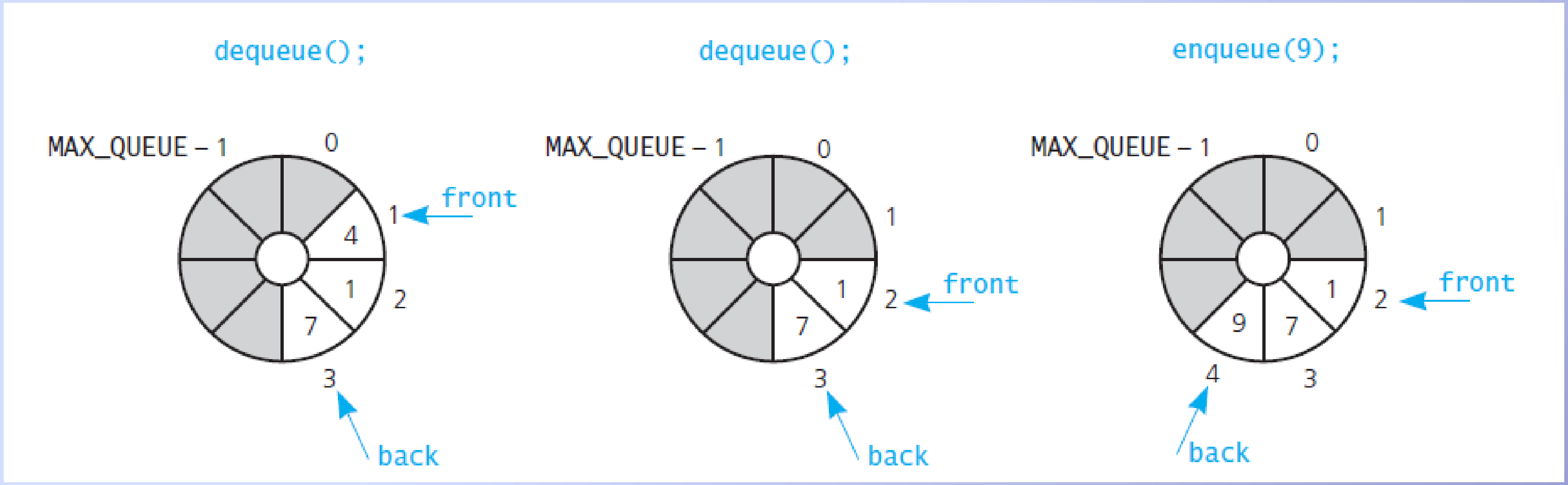


FIGURE 14-9 The effect of three consecutive operations on the queue in Figure 14-8

An Array-Based Implementation

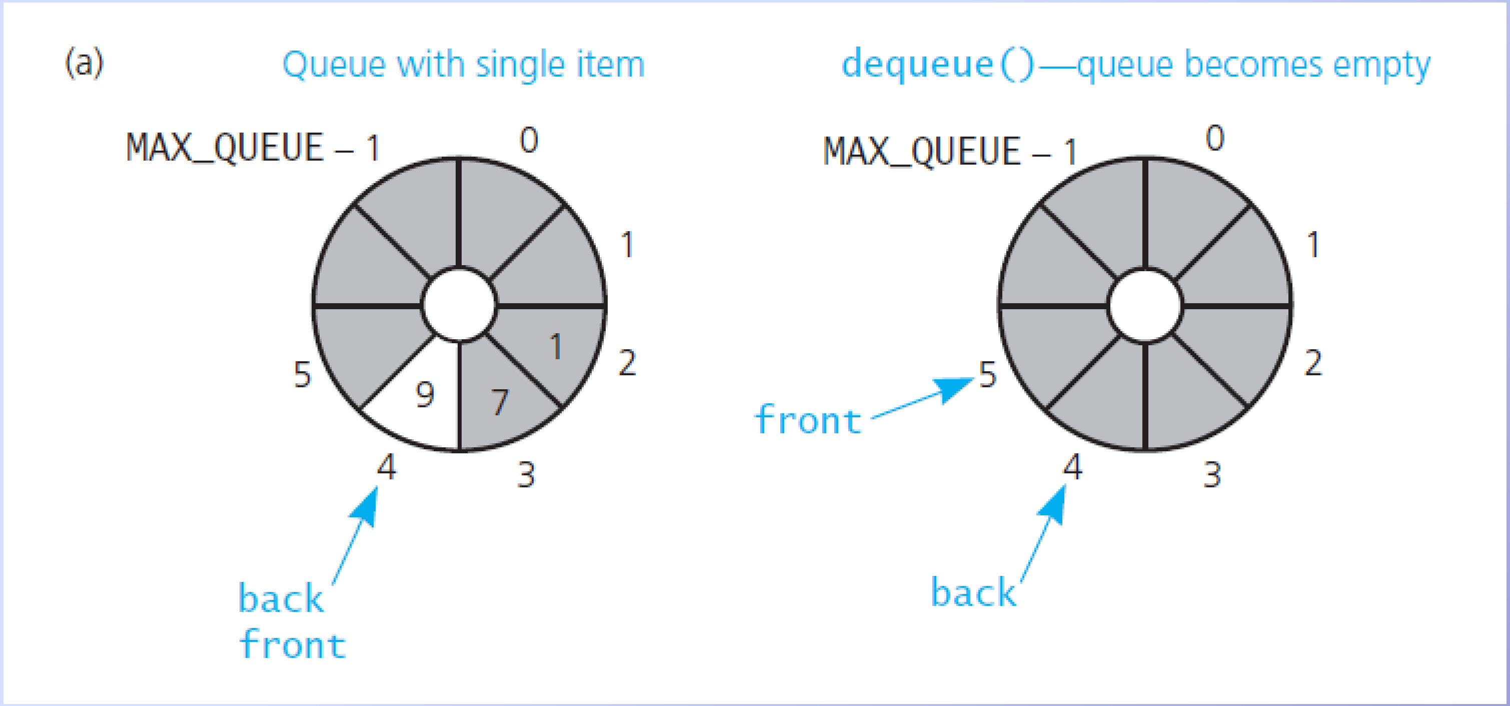


FIGURE 14-10 (a) front passes **`back`** when the queue becomes empty;

An Array-Based Implementation

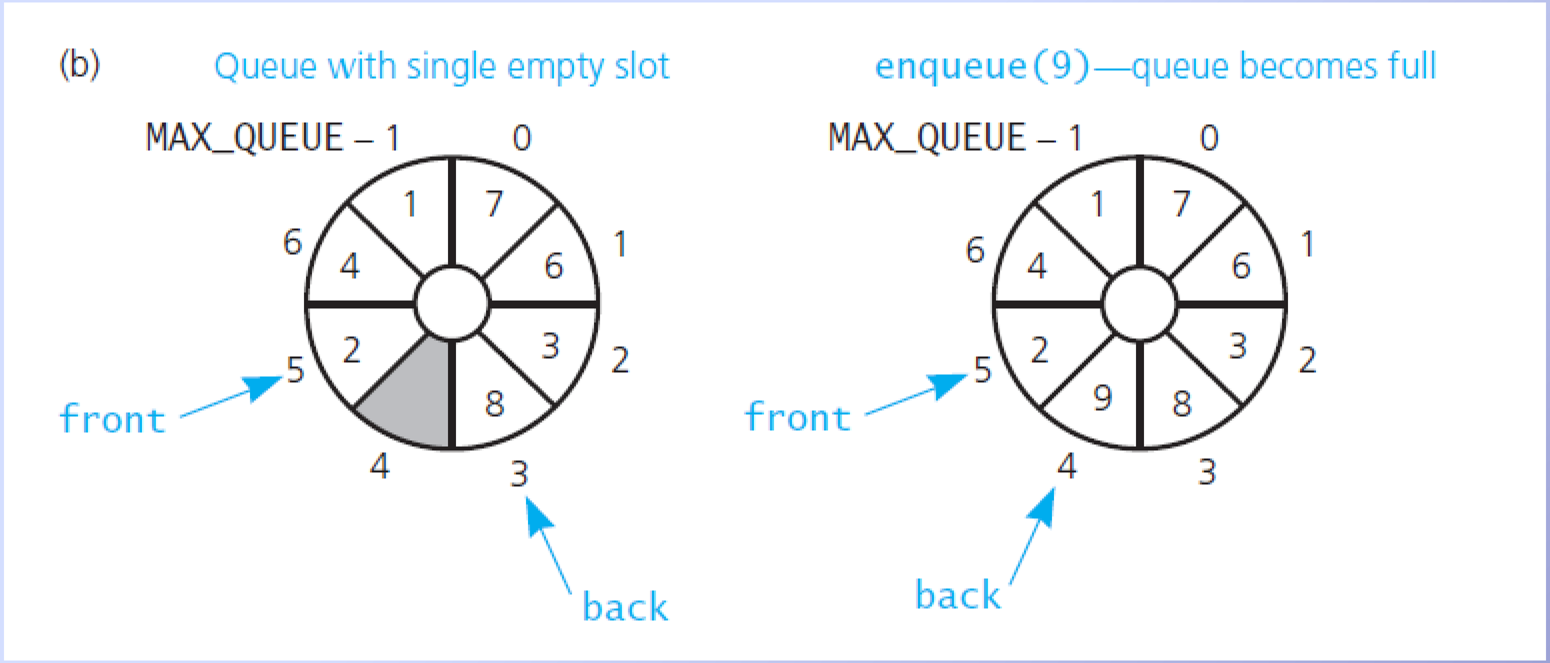


FIGURE 14-10 (b) **`back`** catches up to **`front`** when the queue becomes full

An Array-Based Implementation

- The header file for the class **`ArrayQueue`**, Listing 14-4
- View implementation file, Listing 14-5

Variation of Circular Queue

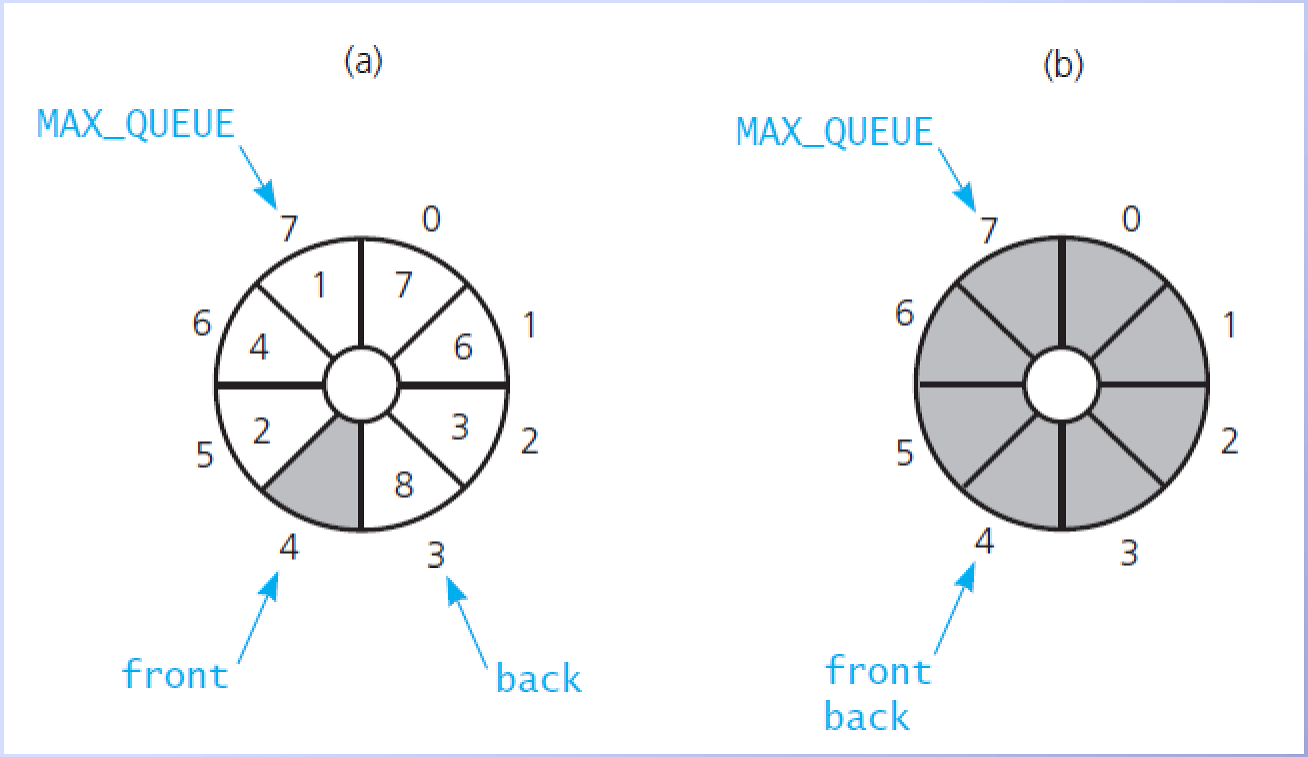
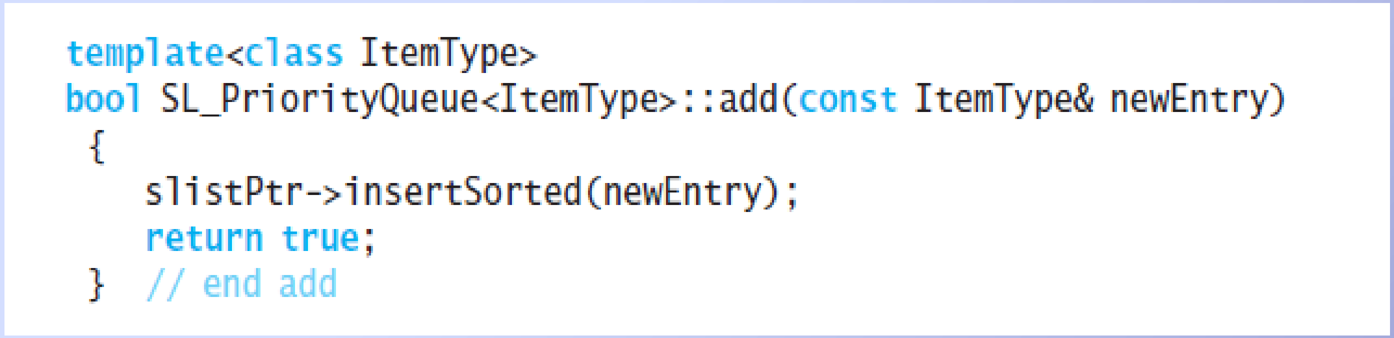


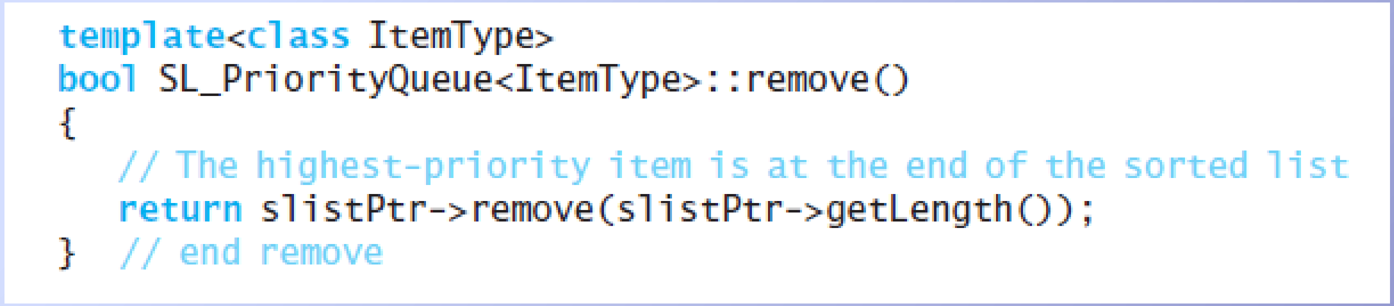
FIGURE 14-11 A more time-efficient circular implementation: (a) a full queue; (b) an empty queue

Implementation of the ADT Priority Queue

- View header file, Listing 14-6
- Note **`add`** and **`remove`** functions



```
template<class ItemType>
bool SL_PriorityQueue<ItemType>::add(const ItemType& newEntry)
 {
    slistPtr->insertSorted(newEntry);
    return true;
 }  // end add
```



```
template<class ItemType>
bool SL_PriorityQueue<ItemType>::remove()
{
   // The highest-priority item is at the end of the sorted list
   return slistPtr->remove(slistPtr->getLength());
}  // end remove
```

End

Chapter 14