 Answers to review questions from Chapter 14

1. What advantages do C++ templates offer designers of generic containers?

**Templates make it possible to define containers of any base type without having to rewrite the code for each type.**

2. When specializing a class template for use as a client, how do you specify what type should be used to fill in the template placeholder?

**You include the name of the type inside angle brackets, as in Vector<int>.**

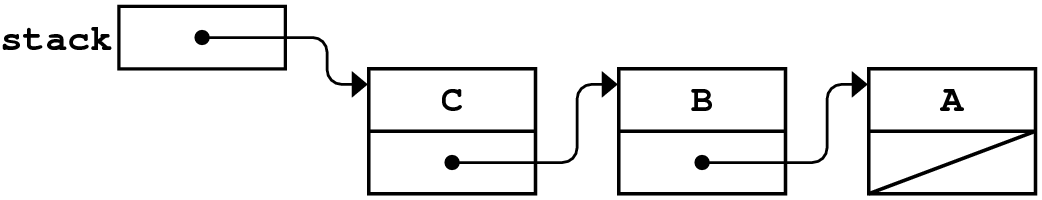
3. Using the linked‑list implementation, draw a diagram of the cells used to represent **myStack** after the following operations have been performed:

Stack<char> myStack;

myStack.push('A');

myStack.push('B');

myStack.push('C');



4. If you use an array to store the underlying elements in a queue, what private instance variables are needed for the **Queue** class?

**The instance variables are the capacity of the array, the index of the head item in the queue, and the index of the next free slot.**

5. What is a *ring buffer?* How does the ring‑buffer concept apply to queues?

**A *ring buffer* is an array in which the ends are logically joined to form a structure that wraps around at the end. The array‑based implementation of queues uses a ring buffer to avoid having to copy data as elements are enqueued and dequeued.**

6. How can you tell whether an array‑based queue is empty? How can you tell whether it has reached its capacity?

**The usual condition for indicating an empty queue is having the head and tail indices match. This representation means that the actual capacity of the queue is one less than the allocated size of the array. The code for enqueue tests the capacity using the following statement:**

if (size() == capacity - 1)

7. Assuming that **INITIAL\_CAPACITY** has the artificially small value of 3, draw a diagram showing the underlying representation of the array‑based queue **myQueue** after the following sequence of operations:

Queue<char> myQueue;

myQueue.enqueue('A');

myQueue.enqueue('B');

myQueue.enqueue('C');

myQueue.dequeue();

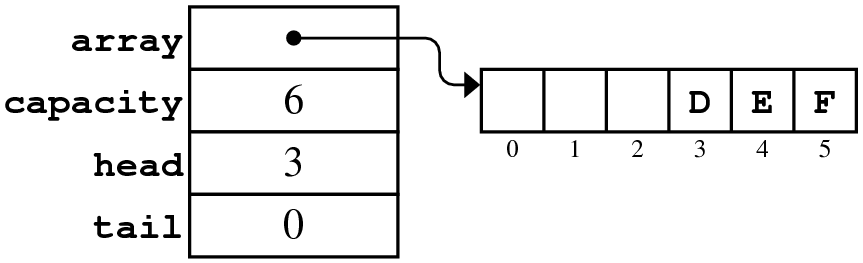
myQueue.dequeue();

myQueue.enqueue('D');

myQueue.enqueue('E');

myQueue.dequeue();

myQueue.enqueue('F');



8. Explain how modular arithmetic is useful in the array‑based implementation of queues.

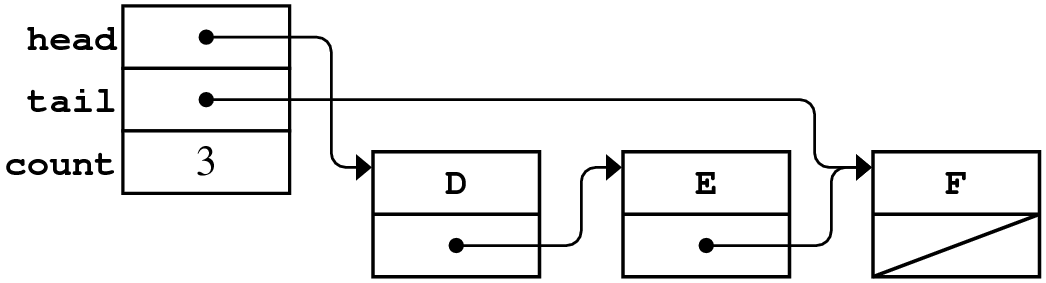
**Modular arithmetic simplifies the calculation of index values in a ring buffer because taking the remainder automatically wraps an index values that is past the end of an array back to the corresponding element at the beginning.**

9. Describe what is wrong with the following implementation of **size** for the array‑based representation of queues:

|  |  |
| --- | --- |
| template <typename ValueType>  int Queue<ValueType>::size() const {  return (tail - head) % capacity;  } | /Users/eroberts/Books/CS2-in-C++/Chapters/01-OverviewOfC++/pictures/Bug/BlueBug.png |

**Subtracting head from tail will often produce a negative value. In C++, the behavior of the remainder operator with negative values depends on the implementation and often produces a negative result.**

10. Draw a diagram showing the internal structure of a linked‑list queue after the computer finishes the set of operations in question 7.



11. How can you tell whether a linked‑list queue is empty?

**A list‑based queue is empty if the head pointer is NULL.**

12. What method do you need to override to redefine bracket selection for a class?

**operator[]**