Problem 1: removeMIN

Write a method called removeMin that takes a stack of integers as a parameter and that removes and returns the smallest value from the stack. For example, if a variable called s stores the following sequence of values:

bottom [2, 8, 3, 19, 7, 3, 2, 42, 9, 3, 2, 7, 12, -8, 4] top

and you make the following call:

int n = removeMin(s);

the method removes and returns the value -8 from the stack, so that the variable n will be -8 after the call and s will store the following values:

bottom [2, 8, 3, 19, 7, 3, 2, 42, 9, 3, 2, 7, 12, 4] top

If the minimum value appears more than once, all occurrences of the minimum should be removed from the stack. For example, given the ending value of the stack above, if we again call removeMin(s), the method would return 2 and would leave the stack in the following state:

bottom [8, 3, 19, 7, 3, 42, 9, 3, 7, 12, 4] top

You are to use one queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively and you must use the Stack and Queue structures.

Problem 2: Range Iterator

Suppose we have a class called IntRange representing a range of integers, implemented as follows:

```
public class IntRange {
private int min;
private int max;   // both ends are inclusive
...
// your iterator class goes here
}
```

Make it possible to use integer ranges as the target of a for-each loop:

```
IntRange range = new IntRange(7, 15);
for (int n : range) {
System.out.print(n + " ");   // 7 8 9 10 11 12 13 14 15
}
```

Write a private class named RangeIterator that would be usable as an inner class to implement an iterator over the integers in an IntRange. Implement the hasNext and next operations; when the remove method is called, you can throw an UnsupportedOperationException. Your iterator should not construct any internal data structures.

```
private class RangeIterator implements Iterator<Integer> {

public RangeIterator() {

// TODO: implement constructor

}

public boolean hasNext() {

// TODO: implement this method

}

public Integer next() {

// TODO: implement this method

}

public void remove() {

throw new UnsupportedOperationException();
}
```

Problem3: isPalindrom

Write a method isPalindrome that takes a Queue of characters as a parameter and that returns whether or not the characters in the queue represent a palindrome (true if they do, false otherwise). A sequence of characters is considered a palindrome if it is the same in reverse order. For example, suppose a Queue called q stores this sequence of values:

front [n,o,o,n] back

Then the following call:

isPalindrome(q)

should return true because this sequence is the same in reverse order. If the list had instead stored these values:

front [a,n,o,o,n] back

the call on isPalindrome would instead return false because this sequence is not the same in reverse order .

The empty queue should be considered a palindrome. You may not make any assumptions about how many elements are in the queue and your method must restore the queue so that it stores the same sequence of values after the call as it did before.

You are to use one stack as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively.

In writing your method, assume that you are using the Stack and Queue interfaces and the ArrayStack and LinkedQueue implementations discussed in lecture.

Problem4: isConsecutive

Write a method isConsecutive that takes a stack of integers as a parameter and that returns whether or not the stack contains a sequence of consecutive integers starting from the bottom of the stack (returning true if it does, returning false if it does not).

Consecutive integers are integers that come one after the other, as in 5, 6, 7, 8, 9, etc. So if a stack s stores the following values:

bottom [3, 4, 5, 6, 7, 8, 9, 10] top

then the call of isConsecutive(s) should return true. If the stack had instead contained this set of values:

bottom [3, 4, 5, 6, 7, 8, 9, 10, 12] top

then the call should return false because the numbers 10 and 12 are not consecutive. Notice that we look at the numbers starting at the bottom of the stack. The following sequence of values would be consecutive except for the fact that it appears in reverse order, so the method would return false:

bottom [3, 2, 1] top

Your method must restore the stack so that it stores the same sequence of values after the call as it did before. Any stack with fewer than two values should be considered consecutive.

You are to use one queue as auxiliary storage to solve this problem. You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like. You also may not solve the problem recursively.