CSE143 Midterm
Winter 2019

Name of Student: _____

Section (e.g., AA):_____ Student Number: _____

The exam is divided into six questions with the following points:

#	Problem Area	Points	Score
1	Recursive Tracing	15	_____
2	Recursive Programming	15	_____
3	ListNodes	15	_____
4	Collections	20	_____
5	Stacks/Queues	25	_____
6	ArrayIntList Programming	10	_____
	Total	100	_____

This is a closed-book/closed-note exam. Space is provided for your answers. There is a "cheat sheet" at the end that you can use as scratch paper. You are not allowed to access any of your own papers during the exam.

The exam is not, in general, graded on style and you do not need to include comments. For the stack/queue and collections questions, however, you are expected to use generics properly and to declare variables using interfaces when possible. You may only use the Stack and Queue methods on the cheat sheet, which are the methods we discussed in class. You are not allowed to use programming constructs like break, continue, or returning from a void method on this exam.

Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks. The only abbreviations you are allowed to use for this exam are:

    S.o.p for System.out.print and S.o.pln for System.out.println

You are NOT to use any electronic devices while taking the test, including calculators. Anyone caught using an electronic device will receive a 10 point penalty.

Do not begin work on this exam until instructed to do so. Any student who starts early or who continues to work after time is called will receive a 10 point penalty.

You are allowed to ask for scratch paper to use as additional space when writing answers, but you must indicate on the original page for the problem that part of the solution is on scratch paper. Failure to do so may result in your work on scratch paper not being graded.

If you finish the exam early, please hand your exam to the instructor and exit quietly through the front door.

Initial here to indicate you have read and agreed to these rules: _____

1. **Recursive Tracing, 15 points:** Consider the following method:

```
public static void mystery(int n, int m) {
    if (n > 0 && m > 0) {
        mystery(n / 10, m / 10);
    }

    if (n % 10 == m % 10) {
        System.out.print("=");
    } else {
        System.out.print("!");
    }
}
```

For each call below, indicate what output is produced:

Method Call	**Output Produced**
mystery(0, 90)	_____
mystery(12, 2)	_____
mystery(12, 22)	_____
mystery(555, 666)	_____
mystery(582, 1522)	_____

2. **Recursive Programming, 15 points:** Write a recursive method called printSequence that takes an integer n as a parameter and that prints exactly n characters composed of * and nested, alternating < >. For example, printSequence(8) should print:

```
<><**><>
```

The output does not include a println. If n is odd, the method should only print one asterisk in the center, as in printSequence(7) which prints:

```
<><*><>
```

Your method should throw an IllegalArgumentException if passed a value less than 1. Below are more examples of calls and the output produced.

Call	Prints
printSequence(1)	*
printSequence(2)	**
printSequence(3)	<*>
printSequence(4)	<**>
printSequence(5)	><*><
printSequence(6)	><**><
printSequence(7)	<><*><>
printSequence(8)	<><**><>

You may not construct any structured objects (no array, String, List, Scanner, etc.). You may not use any loops to solve this problem; you must use recursion.

3. **Linked Lists, 15 points**: Fill in the "code" column in the following table providing a solution that will turn the "before" picture into the "after" picture by modifying links between the nodes shown. You are not allowed to change any existing node's data field value and you are not allowed to construct any new nodes, but you are allowed to declare and use variables of type ListNode (often called "temp" variables). You are limited to at most two variables of type ListNode for each of the four subproblems below.

You are writing code for the ListNode class discussed in lecture:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}
```

As in the lecture examples, all lists are terminated by null and the variables p and q have the value null when they do not point to anything.

before	after	code
p->[1]->[2]->[3] q	p->[1]->[2] q->[3]	
p->[2] q->[1]->[3]	p q->[1]->[2]->[3]	
p->[1]->[2] q->[3]->[4]	p->[2]->[4] q->[3]->[1]	
p->[5]->[4]->[3] q->[2]->[1]	p->[4]->[5]->[2] q->[3]->[1]	

4. **Collections Programming, 20 points:** Write a method called mostCredits that takes a map of student enrollments and a credits map as parameters. The enrollments map uses student names as keys (strings) and has sets of course names as values (also strings). The credits map uses course names as keys (strings) and has the number of credits that course is worth as values (integers). Your method should return the name of the student who is taking the most credits.

For example, a variable called enrollments might contain the following map:

```
{"Kevin"=["CSE143"], "Taylor"=["AMATH301", "CSE311", "PHIL120"],
 "Zaha"=["AMATH301", "CHEM241", "CSE311", "HONORS100"],
 "Jenny"=[], "Alissa"=["CSE143", "MATH308", "AMATH301"]}
```

and a variable called credits might contain the following map:

```
{"AMATH301"=4, "CHEM241"=3, "CSE143"=5, "CSE311"=4, "HONORS100"=1,
 "MATH308"=3, "PHIL120"=5}
```

In this example, the enrollments map indicates that Taylor is taking AMATH 301, CSE 311, and PHIL 120 while Kevin is only taking CSE 143. Also, in this example, the credits map indicates that CSE 311 is a 4-credit class while MATH 308 is a 3-credit class. Suppose that the following call is made:

```
mostCredits(enrollments, credits)
```

This call would return the name "Taylor" because she is taking 13 credits (4 + 4 + 5) which is more than anyone other student; for comparison in this example, Kevin is enrolled for 5 credits, Alissa is enrolled in 12 credits (5 + 3 + 4), and Jenny is enrolled in 0 credits.

If there is a tie between who has the most credits, the student whose name comes alphabetically first should be returned.

For example, consider if we used the previous maps but changed Alissa's classes to

```
["CHEM241", "CSE143", "PHIL120"]
```

Then a call to mostCredits(enrollments, credits) would return "Alissa" because Alissa is taking as many credits as Taylor, but Alissa comes first in the alphabet.

Breaking ties is worth relatively little points for this problem, which means you can earn most of the credit for this problem with a working solution that does not correctly handle the tie case.

If the map of student enrollments is empty, your method should throw an IllegalArgumentException.

You are allowed to create one auxiliary data structure of your choice to solve this problem. You may assume that the given maps are not null and you may assume none of the contents of the maps contain null values. You may also assume that for every class a student is enrolled in there is a corresponding entry for that class in the credits map. The method should not modify the provided map or any of the structures it references.

Use the space on the next page to write your answer. Do not write your solution on this page.

This page is left blank so you have extra space on #4

5. **Stacks/Queues, 25 points:** Write a method named separate that accepts a queue of Strings as a parameter and that rearranges the values by their length. We will assume all of the Strings in the queue are length 1, 2, or 3. The queue should be rearranged so that the strings of length 1 appear first, followed by values that length 2, followed by values that are length 3, and otherwise preserving their relative order in original queue.

For example, suppose a queue called q stored the following sequence of values:

```
front ["cat", "to", "c", "dog", "rat", "b", "am, "a", "d", "hat", "run", "of"] back
```

After the call separate(q), the queue should store the following sequence values (groups underlined for clarity):

```
front ["c", "b", "a", "d", "to", "am", "of", "cat", "dog", "rat", "hat", "run"] back
       |----------------|  |--------------|  |-------------------------------|
            Length 1           Length 2                 Length 3
```

Notice that within the groups, the ordering is the same as the ordering of the original queue.

You may assume that the queue passed to your method is not null and that it does not contain any null values. You may assume all of the Strings in the queue are either length 1, 2, or 3. If the queue is empty, then the sequence of values should be unchanged by the method.

For full credit, obey the following restrictions in your solution. A solution that disobeys them can get partial credit.

* You may use one stack as auxiliary storage. You may not use other structures (arrays, lists, etc.), but you can have as many simple variables as you like.
* Use the Queue interface and Stack/LinkedList classes discussed in class.
* Use stacks/queues in stack/queue-like ways only. Do not use index-based methods such as get, search, or set, or for-each loops or iterators. You may call add, remove, push, pop, peek, isEmpty, and size.
* You may not solve the problem recursively.
* Your solution should run in O(N) time, where N is the size of the queue

You have access to the following two methods and may call them as needed to help you solve the problem:

```
public void s2q(Stack<String> s, Queue<String> q) {
    while (!s.isEmpty()) {
        q.add(s.pop());
    }
}

public void q2s(Queue<String> q, Stack<String> s) {
    while (!q.isEmpty()) {
        s.push(q.remove());
    }
}
```

Use the space on the next page to write your answer. Do not write your solution on this page.

This page is left blank so you have extra space on #5

6. **ArrayIntList Programming, 10 points:** Write a method called repeat that duplicates every element in the list. For example, if a variable called list stores this sequence of values:

```
[1, 3, 2, 7]
```

And you make the following call:

```
list.repeat();
```

then it should store the following values after the call:

```
[1, 1, 3, 3, 2, 2, 7, 7]
```

Notice that it has been doubled in size by repeating each number once.

You are writing a method for the ArrayIntList class discussed in lecture:

```
public class ArrayIntList {
    private int[] elementData; // list of integers
    private int size;          // current # of elements in the list

    <methods>
}
```

You are not to call any other ArrayIntList methods to solve this problem, you are not allowed to define any auxiliary data structures (no array, ArrayList, etc). You may assume that the array has sufficient capacity to store the new values.

To receive full credit, your solution must run in O(n) time, where n is the number of elements in the list. A solution that works but runs in more than O(n) time will receive at most 8/10 points for this question.