Question 1. Consider the following code to initialize the elements of an array:

```java
int [] t = new int[8];
int p = 0;
while ( p < t.length ) {
    t[p] = p * 3;
    p = p + 1;
}
```

a) Suppose that after executing this code, we executed the loop shown below. Show the values that would be associated with each of the elements of the array t after both loops had been executed.

```java
int s = 0;
while ( s < t.length ) {
    t[s] = t[s] + 1;
    s = s + 2;
}
```

b) What would be the final value in count after executing first the loop shown in the introduction to this problem and then the loop shown below (but not the loop discussed in part (a))? (Hint: The use of the % or modulus operator is described in section 7.3.2 of PJSS.)

```java
int count = 0;
int p = 0;
while ( p < t.length ) {
    if ( t[p] % 2 == 0 ) {
        count = count + 1;
    }
    p = p + 1;
}
```

c) Describe the general function of the code shown in part (b). That is, regardless of what values are stored in the array t when the code in part (b) is executed, what relationship will exist between the final value of count and the values in the array?

d) What would be the final value in count after executing first the loop shown in the introduction to this problem and then the loop shown below (but not the loops discussed in part (a))? (Hint: The use of the % or modulus operator is described in section 7.3.2 of PJSS.)

```java
int count = 0;
int p = 0;
while ( p < t.length ) {
    if ( p % 2 == 0 ) {
        count = count + 1;
    }
    p = p + 1;
}
```

e) Show how to re-write the loop shown in part (a) as a for loop.

Question 2. Consider the imaginary IP network fragment show in Figure 1. The network is composed of three physical networks connected to one another by the internal router shown in the center of the diagram. To make it clear that they are distinct physical networks, the diagram shows that each network uses a very distinct networking technology: the two laptops are shown as part of a wireless network, the two cartoon computers are attached to an Ethernet, and the connection between the internal router in the center of the diagram and the external router that
connects to the rest of the Internet is drawn as a ring network. The details of ring, wireless and Ethernet technologies, however, are irrelevant to the problem. All that matters is that there are three distinct physical networks involved. Since the internal router has connections to all three physical networks, it has three distinct IP addresses — one for each of the interfaces through which it connects to the three physical networks. The IP addresses associated with router interfaces and with the computers connected to the Ethernet and wireless networks are shown in Figure 1. The physical addresses associated with all of the IP addresses shown in the figure are given in the following table:

<table>
<thead>
<tr>
<th>IP Address</th>
<th>Physical Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.134.172.12</td>
<td>02:4F:30:A5:00:1F</td>
</tr>
<tr>
<td>200.134.172.9</td>
<td>39:22:0D:10:3B:33</td>
</tr>
<tr>
<td>137.165.2.1</td>
<td>22:31:FF:21:4B:18</td>
</tr>
<tr>
<td>137.165.4.201</td>
<td>02:4F:30:03:F2:11</td>
</tr>
<tr>
<td>200.134.73.1</td>
<td>39:22:0D:93:83:AA</td>
</tr>
<tr>
<td>200.134.73.19</td>
<td>20:90:33:D0:D6:41</td>
</tr>
<tr>
<td>200.134.73.6</td>
<td>50:57:30:28:01:9B</td>
</tr>
</tbody>
</table>

(a) Any IP packet traveling through this network will include source and destination addresses in the header of the hardware network packet (Ethernet, WiFi, or ring) and source and destination addresses in the IP packet encapsulated in the data area of the hardware packet. Thus, each packet would contain a total of 4 addresses. Would it be possible for a valid packet observed within this collection of networks to contain the addresses 200.134.172.12, 200.134.172.9, 02:4F:30:A5:00:1F and 39:22:0D:10:3B:33? If not, explain why. If so, on which of the three networks shown in the figure (Ethernet, WiFi, or ring) could this packet be observed?

(b) Would it be possible for a valid packet observed within this collection of networks to contain the addresses 200.134.172.12, 200.134.73.6, 02:4F:30:A5:00:1F and 50:57:30:28:01:9B? If not, explain why. If so, on which of the three networks shown in the figure could this packet be observed?

(c) Would it be possible for a valid packet observed within this collection of networks to contain the addresses 200.134.172.12, 200.134.73.6, 02:4F:30:A5:00:1F and 01:3F:22:9B:6C:99? If not, explain why. If so, on which of the three networks shown in the figure could this packet be observed?
(d) Suppose 200.134.172.12 wants to send an IP packet to 200.134.73.19. How would 200.134.172.12 determine whether the IP packet should be sent to 200.134.73.19 directly or via the router? Once it determined it should send the packet to the router, how would 200.134.172.12 determine the router’s IP address? How would Station 200.134.172.12 determine the router’s physical address?

(e) Suppose that a packet originally sent from the machine with IP address 200.134.73.6 to 202.34.190.4 was observed while traveling through the ring network between the two routers shown in the figure. What would be the IP source and destination addresses and the physical source and destination addresses in the packet?

Question 3. The code in Figure 2 defines a GradeCollection class that contains a toString method definition and several other useful, but very poorly named, methods. Suppose that we have executed the statements:

```
GradeCollection g = new GradeCollection();
g = new GradeCollection( "Saud", 92, g );
g = new GradeCollection( "Amelia", 95, g );
g = new GradeCollection( "Alex", 97, g );
g = new GradeCollection( "Garett", 94, g );
```

Indicate the String or int value that would be produced by each of the following expressions. Also provide a brief description of what the method invoked in each expression does in general to a GradeCollection.

a) g.toString()
b) g.ys()
c) g.ma()

d) The definition of toString in this class is a bit more complicated that the toString methods you wrote or used in lab. It has an extra case. An alternate version of toString revised to use only two cases (and in other ways) is shown below:

```
public String toString() {
    if (empty) {
        return ".";
    } else {
        return rest.toString() + "," + studentName + ": " + studentScore ;
    }
}
```

This new version will not actually work the same as the original definition. Show the output that would be produced for subpart (a) of this question if toString’s definition was revised in this way.

e) In all of the recursive collection classes we have seen, the class has included two constructors that functioned in very predictable ways. One took no parameters and just set empty to true. The other set empty to false and copied parameter values to instance variables. We are free, however, to place other forms of code in the constructor. Consider the alternative to the “non-empty” constructor for the GradeCollection class shown below:

```
public GradeCollection( String student, int score, GradeCollection existing) {
    if ( !existing.empty && score < existing.studentScore) {
        studentName = existing.studentName;
        studentScore = existing.studentScore;
        rest = new GradeCollection( student, score, existing.rest );
    } else {
        studentName = student;
        studentScore = score;
        rest = existing;
    }
}
```

How would the answer to part (a) change if we replaced the constructor shown in Figure 2 with this new version?
public class GradeCollection {
    private String studentName;
    private int studentScore;
    private GradeCollection rest;
    private boolean empty = false;

    public GradeCollection() {
        empty = true;
    }

    public GradeCollection( String student, int score, GradeCollection existing) {
        studentName = student;
        studentScore = score;
        rest = existing;
    }

    public int cbt() {
        if (empty) {
            return 0;
        } else {
            return studentScore + rest.cbt();
        }
    }

    public int ys() {
        if (empty) {
            return 0;
        } else {
            return 1 + rest.ys();
        }
    }

    public int ma() {
        if (empty) {
            return 0;
        } else { 
            return cbt()/ys();
        }
    }

    public String toString() {
        if (empty) {
            return "";
        } else if (rest.empty) {
            return studentName + ": " + studentScore + ".";
        } else {
            return studentName + ": " + studentScore + "," + rest.toString();
        }
    }
}

Figure 2: Code for GradeCollection class