Question 1. Consider the following set of class definitions involving inheritance:

```java
public class Child {
    public Child() {
    }

    public int descendants() {
        return children();
    }

    public int children() { return 0; }
}

public class Parent extends Child {
    private final int MAX_FAMILY = 70;
    private int familySize = 0;
    private Child[] children = new Child[MAX_FAMILY];

    public Parent() {
    }

    public void addChild(Child newBorn) {
        if (familySize < MAX_FAMILY) {
            children[familySize] = newBorn;
            familySize++;
        }
    }

    public int children() { return familySize; }

    public Child getChild(int birthOrder) {
        if (birthOrder < familySize) {
            return children[birthOrder];
        } else {
            return null;
        }
    }
}

public class GrandParent extends Parent {
    public GrandParent() {
    }

    public int descendants() {
        int total = 0;
        for (int r = 0; r < children(); r++) {
            total = total + getChild(r).descendants() + 1;
        }
        return total;
    }
```
Suppose that the following code using this classes has been executed in some method:

```java
Child john2 = new Child();
Child jim = new Child();
Parent joan = new Parent();
joan.addChild(jim);
joan.addChild(john2);
Child johnjr = new Child();
GrandParent mary = new GrandParent();
mary.addChild(joan);
mary.addChild(johnjr);
```

a) What value would be produced by the invocation

```
mary.descendants()
```

if it occurred after the code shown was executed?

b) Would the assignment

```
Child newKid = mary;
```

be legal in this context. If so, what value would be produced by the invocation

```
newKid.descendants()
```

if it occurred after the code shown was executed?

c) Would the assignment

```
GrandParent grandMa = joan;
```

be legal in this context. If so, what value would be produced by the invocation

```
grandMa.descendants()
```

if it occurred after the code shown was executed?

**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:

(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 200.134.73.19, 200.134.172.9, 20:90:33:D0:D6:41 and 39:22:0D:10:3B:33? If not, explain why. If so, on which of the three networks shown in the figure could this packet be observed?
Figure 1: An IP Network.

<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>

(b) Would it be possible for a valid packet observed within this collection of networks to contain the addresses 200.134.73.19, 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 (Ethernet, WiFi, or ring) 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 200.134.73.19 to 182.33.19.7 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.

Table 1 shows the results of executing the main loop of Dijkstra’s shortest path algorithm several times to build a forwarding table for the router named A. Unfortunately, the table is incomplete and much of the information about the neighbors of the routers disappeared during a freak transcription accident involving cod liver oil and a small rodent.

Please follow the steps of Dijkstra’s Algorithm to complete the entries in the Best Route Length and First Step columns of the table. The rows in which the connections to neighboring routers rows are completely empty are the
places where the information disappeared. You should leave these table entries blank. You will discover you do not need this information to complete the Best Route Length and First Step columns of the table if you simply follow algorithm 1 as described below. Assume that all edges have positive integer lengths.

Table 1: Shortest Path Table

<table>
<thead>
<tr>
<th>Cities</th>
<th>Best Route Length</th>
<th>First Step</th>
<th>Status</th>
<th>Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>-</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>C</td>
<td>Adjacent</td>
<td>K: 1   H: 6 D: 5 J:2</td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>C</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td>L</td>
<td>Adjacent</td>
<td>B: 1   H: 2</td>
</tr>
<tr>
<td>E</td>
<td>7</td>
<td>E</td>
<td>Adjacent</td>
<td>C: 3   L: 3 H: 1 D: 2</td>
</tr>
<tr>
<td>F</td>
<td>-</td>
<td></td>
<td>Distant</td>
<td>L: 3   J: 2</td>
</tr>
<tr>
<td>G</td>
<td>6</td>
<td>L</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>10</td>
<td>C</td>
<td>Adjacent</td>
<td>I: 1   D: 2</td>
</tr>
<tr>
<td>I</td>
<td>10</td>
<td>C</td>
<td>Adjacent</td>
<td>K: 3   H: 6 D: 5 G: 3</td>
</tr>
<tr>
<td>J</td>
<td>5</td>
<td>C</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>14</td>
<td>C</td>
<td>Adjacent</td>
<td>F: 1   C: 3 L: 2</td>
</tr>
<tr>
<td>L</td>
<td>4</td>
<td>L</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>7</td>
<td>C</td>
<td>Known</td>
<td></td>
</tr>
</tbody>
</table>

**Algorithm 1** DIJKSTRA'S SHORTESTPATH

1: Mark starting point as *KNOWN* with length 0
2: Identify each neighbor of start as *ADJACENT*
3: Set first step of each neighbor of start to itself
4: Set route length of each neighbor to first step distance
5: While you don’t know the shortest path to all the cities:
   6: Select adjacent city with shortest 'best route' value
   7: Identify that adjacent city with shortest best route as *KNOWN*
   8: Mark neighbors of new *KNOWN* city that were *DISTANT* as *ADJACENT*
   9: Update path lengths and record first steps to *ADJACENT* neighbors of new *KNOWN* city.