CSE 143, Winter 2011
Final Exam Key

1.

Statement	Output
`var1.a();`	`Vegetable a`
`var1.b();`	`Computer b / Mineral b / Vegetable a`
`var1.c();`	`error`
`var2.a();`	`Animal a`
`var2.b();`	`Mineral b / Animal a`
`var3.a();`	`Vegetable a`
`var3.b();`	`Mineral b / Vegetable a`
`var4.a();`	`error`
`((Computer) var1).b();`	`Computer b / Mineral b / Vegetable a`
`((Computer) var1).c();`	`Computer c`
`((Computer) var2).c();`	`error`
`((Animal) var2).c();`	`Mineral b / Animal a / Animal c`
`((Computer) var3).b();`	`error`
`((Vegetable) var4).b();`	`Mineral b / Vegetable a`
`((Animal) var4).c();`	`error`

2.

```
public class GradStudent extends Student implements Comparable<GradStudent> {
    private String advisor;

    public GradStudent(String name, int year, String advisor) {
        super(name, year + 4);
        this.advisor = advisor;
    }

    public void addCourse(String name) {
        if (getCourseCount() < 3) {
            super.addCourse(name);
        }
    }

    public String getAdvisor() {
        return advisor;
    }

    public double getTuition() {
        if (getCourseCount() <= 1) {
            return 500.0;
        } else {
            return 2 * super.getTuition();
        }
    }

    public boolean isBurntOut() {
        return getYear() >= 9 || getCourseCount() >= 3;
    }

    public int compareTo(GradStudent other) {
        if (getYear() != other.getYear()) {
            return getYear() - other.getYear();
        } else if (getCourseCount() != other.getCourseCount()) {
            return getCourseCount() - other.getCourseCount();
        } else {
            return advisor.compareTo(other.getAdvisor());
        }
    }
}
```

```
public class GradStudent extends Student implements Comparable<GradStudent> {
    private String advisor;

    public GradStudent(String name, int year, String advisor) {
        super(name, year);
        this.advisor = advisor;
    }

    public void addCourse(String name) {
        if (getCourseCount() < 3) {
            super.addCourse(name);
        }
    }

    public String getAdvisor() {
        return advisor;
    }

    public double getTuition() {
        if (getCourseCount() >= 2) {
            return 2 * super.getTuition();
        } else {
            return 500.0;
        }
    }

    public int getYear() {
        return super.getYear() + 4;
    }

    public boolean isBurntOut() {
        if (getYear() >= 9) {
            return true;
        } else if (getYear() < 9 && getCourseCount() >= 3) {
            return true;
        } else {
            return false;
        }
    }

    public int compareTo(GradStudent other) {
        if (getYear() > other.getYear()) {
            return 1;
        } else if (getYear() < other.getYear()) {
            return -1;
        } else if (getCourseCount() > other.getCourseCount()) {
            return 1;
        } else if (getCourseCount() < other.getCourseCount()) {
            return -1;
        } else if (advisor.compareTo(other.getAdvisor()) > 0) {
            return 1;
        } else if (advisor.compareTo(other.getAdvisor()) < 0) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

3.

```
public void removeEvens() {
    if (front == null) {
        front = new ListNode(0);
    } else {
        int removed = 0;

        // remove from front as many as necessary
        while (front != null && front.data % 2 == 0) {
            front = front.next;
            removed++;
        }

        if (front == null) {
            front = new ListNode(removed);
        } else {
            // remove from the rest
            ListNode current = front;
            while (current.next != null) {
                if (current.next.data % 2 == 0) {
                    current.next = current.next.next;
                    removed++;
                } else {
                    current = current.next;
                }
            }

            // append number of elements removed at end
            current.next = new ListNode(removed);
        }
    }
}



// recursive solution (isn't it pretty?!)
public void removeEvens() {
    front = removeEvens(front, 0);
}

private ListNode removeEvens(ListNode current, int removed) {
    if (current == null) {
        current = new ListNode(removed);
    } else if (current.data % 2 == 0) {
        current = removeEvens(current.next, removed + 1);
    } else {
        current.next = removeEvens(current.next, removed);
    }
    return current;
}
```

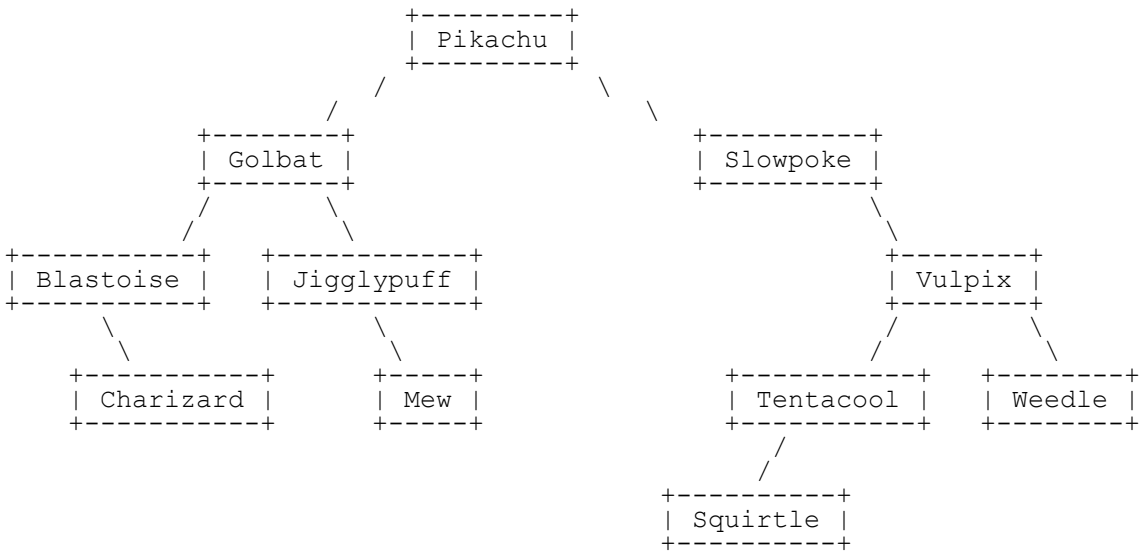
4.

(a) Indexes examined: **7, 3, 5, 6** Value returned: **-7**

(b)
```
{29, 17,  3, 94, 46,  8, -4, 12}
{-4, 17,  3, 94, 46,  8, 29, 12}
{-4,  3, 17, 94, 46,  8, 29, 12}
{-4,  3,  8, 94, 46, 17, 29, 12}
```

(c)
```
{29,  17,    3,  94,   46,   8,  -4,  12}
{29,  17,    3,  94}  {46,   8,  -4,  12}     split
{29,  17}  { 3,  94}  {46,   8} {-4,  12}     split
{29} {17}  { 3} {94}  {46} { 8} {-4} {12}     split
{17,  29}  { 3,  94}  { 8,  46} {-4,  12}     merge
{3,   17,   29,  94}  {-4,   8,  12,  46}     merge
{-4,   3,    8,  12,   17,  29,  46,  94}     merge
```

5. (a)



(b)

Pre: Pikachu, Golbat, Blastoise, Charizard, Jigglypuff, Mew, Slowpoke, Vulpix, Tentacool, Squirtle, Weedle

In: Blastoise, Charizard, Golbat, Jigglypuff, Mew, Pikachu, Slowpoke, Squirtle, Tentacool, Vulpix, Weedle

Post: Charizard, Blastoise, Mew, Jigglypuff, Golbat, Squirtle, Tentacool, Weedle, Vulpix, Slowpoke, Pikachu

6.

```
// "counting up" solution
public int countBelow(int minLevel) {
    if (minLevel <= 0) {
        throw new IllegalArgumentException();
    } else {
        return countBelow(overallRoot, 1, minLevel);
    }
}

private int countBelow(IntTreeNode node, int currentLevel, int minLevel) {
    if (node == null) {
        return 0;
    } else if (currentLevel < minLevel) {
        return countBelow(node.left,  currentLevel + 1, minLevel)
             + countBelow(node.right, currentLevel + 1, minLevel);
    } else {
        return 1 + countBelow(node.left,  currentLevel + 1, minLevel)
                 + countBelow(node.right, currentLevel + 1, minLevel);
    }
}


// "counting down" solution
public int countBelow(int min) {
    if (min <= 0) {
        throw new IllegalArgumentException();
    } else {
        return countBelow(overallRoot, min);
    }
}

private int countBelow(IntTreeNode node, int min) {
    int count = 0;
    if (node != null) {
        if (min <= 1) {
            count++;
        }
        count += countBelow(node.left, min - 1) + countBelow(node.right, min - 1);
    }
    return count;
}



// "pass a count parameter" solution
public int countBelow(int minLevel) {
    if (minLevel <= 0) {
        throw new IllegalArgumentException();
    } else {
        return countBelow(overallRoot, 1, minLevel, 0);
    }
}

private int countBelow(IntTreeNode node, int currentLevel, int minLevel, int count) {
    if (node != null) {
        if (currentLevel >= minLevel) {
            count++;
        }
        count = countBelow(node.left,  currentLevel + 1, minLevel, count);
        count = countBelow(node.right, currentLevel + 1, minLevel, count);
    }
    return count;
}
```

7.

```
public void removeMatchingLeaves(IntTree other) {
    overallRoot = removeMatchingLeaves(overallRoot, other.overallRoot);
}

private IntTreeNode removeMatchingLeaves(IntTreeNode node, IntTreeNode otherNode) {
    if (node != null && otherNode != null) {
        if (node.left == null && node.right == null && node.data == otherNode.data) {
            node = null;
        } else {
            node.left = removeMatchingLeaves(node.left, otherNode.left);
            node.right = removeMatchingLeaves(node.right, otherNode.right);
        }
    }
    return node;
}
```

```
public void removeMatchingLeaves(IntTree other) {
    overallRoot = removeMatchingLeaves(overallRoot, other.overallRoot);
}

private IntTreeNode removeMatchingLeaves(IntTreeNode node, IntTreeNode otherNode) {
    if (node == null) {
        return null;
    } else if (otherNode == null) {
        return node;
    } else {
        node.left = removeMatchingLeaves(node.left, otherNode.left);
        node.right = removeMatchingLeaves(node.right, otherNode.right);

        if (node.left != null && node.right != null) {
            if (node.data == otherNode.data) {
                return null;
            }
        }
    }
    return node;
}
```