Solutions to Midterm Exam

Although I’m not quite sure of the reason, this midterm exam didn’t go as well as midterms have in recent years. The median was 70 percent, which is substantially lower than what I shoot for and lower than it has been in recent years. Part of the problem seems to center on problem 3, which seemed to throw many of you for a loop, but even so I am concerned that more of you are struggling with the algorithmic aspects of the material than is usually the case.

The complete histogram of grades appears below. You can determine your letter grade—which I’ve curved so that the median is the middle of the B range—by looking up the percentage score (the score written at the top of your exam) in the table at the upper left corner of the histogram.

<table>
<thead>
<tr>
<th>Range</th>
<th>Grade</th>
<th>N</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>98–100</td>
<td>A+</td>
<td>13</td>
<td>64</td>
<td>69</td>
<td>79</td>
<td>84</td>
<td>64</td>
<td>69</td>
<td>79</td>
<td>84</td>
<td>64</td>
<td>69</td>
</tr>
<tr>
<td>88–97</td>
<td>A</td>
<td>25</td>
<td>54</td>
<td>59</td>
<td>64</td>
<td>68</td>
<td>74</td>
<td>78</td>
<td>83</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>83–87</td>
<td>A–</td>
<td>27</td>
<td>54</td>
<td>59</td>
<td>64</td>
<td>68</td>
<td>74</td>
<td>78</td>
<td>83</td>
<td>89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>75–82</td>
<td>B+</td>
<td>29</td>
<td>54</td>
<td>59</td>
<td>63</td>
<td>68</td>
<td>74</td>
<td>78</td>
<td>83</td>
<td>89</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>66–74</td>
<td>B</td>
<td>33</td>
<td>53</td>
<td>59</td>
<td>63</td>
<td>68</td>
<td>73</td>
<td>78</td>
<td>83</td>
<td>88</td>
<td>93</td>
<td>98</td>
</tr>
<tr>
<td>61–65</td>
<td>B–</td>
<td>26</td>
<td>53</td>
<td>59</td>
<td>63</td>
<td>67</td>
<td>73</td>
<td>78</td>
<td>83</td>
<td>88</td>
<td>93</td>
<td>98</td>
</tr>
<tr>
<td>56–60</td>
<td>C+</td>
<td>15</td>
<td>53</td>
<td>58</td>
<td>63</td>
<td>67</td>
<td>73</td>
<td>77</td>
<td>83</td>
<td>87</td>
<td>93</td>
<td>98</td>
</tr>
<tr>
<td>50–55</td>
<td>C</td>
<td>21</td>
<td>53</td>
<td>58</td>
<td>63</td>
<td>67</td>
<td>73</td>
<td>77</td>
<td>83</td>
<td>87</td>
<td>93</td>
<td>98</td>
</tr>
<tr>
<td>45–49</td>
<td>C–</td>
<td>6</td>
<td>53</td>
<td>58</td>
<td>62</td>
<td>66</td>
<td>73</td>
<td>77</td>
<td>83</td>
<td>87</td>
<td>93</td>
<td>98</td>
</tr>
<tr>
<td>35–44</td>
<td>D</td>
<td>14</td>
<td>43</td>
<td>53</td>
<td>57</td>
<td>62</td>
<td>66</td>
<td>72</td>
<td>77</td>
<td>83</td>
<td>87</td>
<td>92</td>
</tr>
<tr>
<td>00–34</td>
<td>NP</td>
<td>4</td>
<td>43</td>
<td>53</td>
<td>57</td>
<td>62</td>
<td>66</td>
<td>72</td>
<td>77</td>
<td>82</td>
<td>87</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>43</td>
<td>53</td>
<td>57</td>
<td>61</td>
<td>66</td>
<td>72</td>
<td>76</td>
<td>82</td>
<td>87</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>42</td>
<td>49</td>
<td>53</td>
<td>56</td>
<td>61</td>
<td>65</td>
<td>70</td>
<td>75</td>
<td>82</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>39</td>
<td>41</td>
<td>47</td>
<td>52</td>
<td>56</td>
<td>61</td>
<td>65</td>
<td>70</td>
<td>75</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td>41</td>
<td>46</td>
<td>50</td>
<td>55</td>
<td>61</td>
<td>65</td>
<td>70</td>
<td>75</td>
<td>82</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>38</td>
<td>40</td>
<td>46</td>
<td>50</td>
<td>55</td>
<td>61</td>
<td>65</td>
<td>70</td>
<td>75</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>29</td>
<td>36</td>
<td>40</td>
<td>46</td>
<td>50</td>
<td>55</td>
<td>61</td>
<td>65</td>
<td>70</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>23</td>
<td>26</td>
<td>30</td>
<td>35</td>
<td>40</td>
<td>45</td>
<td>50</td>
<td>55</td>
<td>60</td>
<td>65</td>
</tr>
</tbody>
</table>
Problem 1: Karel the Robot (10 points)

```java
/*
 * File: KarelCare
 * ---------------
 * Karel looks through the hospital ward for patients with
 * temperatures over 100 and paints the square under the
 * temperature red so that doctors can treat the patient.
 */
import stanford.karel.*;
public class KarelCare extends SuperKarel {

/*
 * Runs the program. This code find every bed in the ward and
 * then calls checkTemperature. This code handles the final
 * column correctly, although that was not required.
 */
    public void run() {  
        while (frontIsClear()) {
            if (beepersPresent()) {
                checkTemperature();
            }  
            move();
        }
        if (beepersPresent()) {
            checkTemperature();
        }
    }

/*
 * Flags any temperatures greater than 100. This code operates
 * by taking away 100 beepers (if possible) and seeing whether
 * there are any left. It then puts all the beepers back.
 */
    private void checkTemperature() {
        for (int i = 0; i < 100; i++) {
            if (beepersPresent()) {
                pickBeeper();
            }
        }
        if (beepersPresent()) {
            paintCorner(RED);
        }
        while (beepersInBag()) {
            putBeeper();
        }
    }
}
```

On the evening exam, the color of the corner was **YELLOW**.
Problem 2: Simple Java programs (10 points)

(2a) Afternoon exam:

\[
\begin{align*}
6 / 5 + 5 + 8 \mod 3 &= 8 & \text{true} \\
('D' - 'A') + '0' &= 3 \text{ or } 51 \\
1 + 2 + "3" + 4 + 5 &= "3345"
\end{align*}
\]

Evening exam:

\[
\begin{align*}
6 / 5 + 5 + 8 \mod 3 &= 8 & \text{false} \\
('6' - '2') + 'A' &= 'E' \text{ or } 69 \\
1 * 2 + "3" + 4 * 5 &= "2320"
\end{align*}
\]

(2b) "lollipop"

(2c) \[
\begin{array}{|c|c|c|}
\hline
\text{afternoon exam} & \text{evening exam} \\
\hline
\text{valentine: str = XOXOXOXOXO, n = 0} & \text{valentine: str = OXOXOXOXOX, n = 0} \\
\text{heart: str = XO, n = 5} & \text{heart: str = OX, n = 5} \\
\text{XOXOX} & \text{OXOXO} \\
\hline
\end{array}
\]
Problem 3: Simple Java programs (15 points)

```java
/*
 * File: TotientTable.java
 * -----------------------
 * This program produces a table of values of the Euler totient
 * function from 2 up to the maximum specified by the constant
 * LIMIT. The totient function plays a central role in the
 * mathematics behind the RSA public encryption algorithm.
 */

import acm.program.*;

public class TotientTable extends ConsoleProgram {

    /* Creates the table */
    public void run() {
        for (int i = 2; i <= LIMIT; i++) {
            println("totient(" + i + ") = " + totient(i));
        }
    }

    /* Calculates the Euler totient function of n. The totient is
    * the number of integers between 1 and n that share no factors
    * with n (other than 1).
    */
    private int totient(int n) {
        int count = 0;
        for (int i = 1; i < n; i++) {
            if (!shareACommonFactor(i, n)) count++;
        }
        return count;
    }

    /* Returns true if n1 and n2 share a common factor (other than 1).
    * In mathematics, such numbers are said to be relatively prime.
    */
    private boolean shareACommonFactor(int n1, int n2) {
        for (int i = 2; i <= Math.min(n1, n2); i++) {
            if (n1 % i == 0 && n2 % i == 0) return true;
        }
        return false;
    }

    /* Private constants */
    private static final int LIMIT = 12;
}
```

An even simpler strategy would be to use the `gcd` function that was introduced in the discussion of rational numbers and appears on page 202 of the text. In this case, you could leave out the definition of `shareACommonFactor` and change the `if` in `totient` to

```java
if (gcd(i, n) == 1) count++;
```

On the evening exam, the constant `LIMIT` was renamed to `MAX`. 
Problem 4: Using the graphics and random number libraries (15 points)

There are several approaches for solving this problem, of which two are shown here. The first stores the direction of motion as one of the constants NORTH, EAST, SOUTH, or WEST.

```java
/*
 * File: Snake.java
 * ---------------
 * This program plays the simplified Snake game from the midterm exam.
 */

import acm.graphics.*;
import acm.program.*;
import java.awt.event.*;

public class Snake extends GraphicsProgram {

    /* Initializes the graphics state */
    public void init() {
        x = getWidth() / 2;
        y = getHeight() / 2;
        dir = EAST;
        addMouseListeners();
    }

    /* Runs the simulation */
    public void run() {
        while (getElementAt(x, y) == null) {
            drawSquare();
            updateSnakeHeadLocation();
            pause(PAUSE_TIME);
        }
    }

    /* Adds a square to the window centered at (x, y) */
    private void drawSquare() {
        GRect rect = new GRect(SQUARE_SIZE, SQUARE_SIZE);
        rect.setFilled(true);
        add(rect, x - SQUARE_SIZE / 2, y - SQUARE_SIZE /2);
    }

    /* Updates the location of the snake head */
    private void updateSnakeHeadLocation() {
        switch (dir) {
            case NORTH: y -= SQUARE_SIZE; break;
            case EAST:  x += SQUARE_SIZE; break;
            case SOUTH: y += SQUARE_SIZE; break;
            case WEST:  x -= SQUARE_SIZE; break;
        }
    }
}
```
/* Reacts to a mouse-pressed event */
public void mousePressed(MouseEvent e) {
    if (dir == NORTH || dir == SOUTH) {
        dir = (e.getX() > x) ? EAST : WEST;
    } else {
        dir = (e.getY() > y) ? SOUTH : NORTH;
    }
}

/* Instance variables */
private double x;      /* The x coordinate of the snake head */
private double y;      /* The y coordinate of the snake head */
private int dir;       /* The direction the head is moving */

/* Private constants */
private static final int NORTH = 0;
private static final int EAST = 1;
private static final int SOUTH = 2;
private static final int WEST = 3;
private static final int PAUSE_TIME = 100;
private static final double SQUARE_SIZE = 15;

The second version is similar except that it keeps track of two independent velocity components (dx and dy) in much the same way that Breakout did.

/* File: SnakeUsingDeltas.java
 * ---------------------------
 * This program plays the simplified Snake game from the midterm exam.
 * This version does not use an enumerated direction type but instead
 * keeps track of separate dx and dy values;
 * */
import acm.graphics.*;
import acm.program.*;
import java.awt.event.*;

public class SnakeUsingDeltas extends GraphicsProgram{

    /* Initializes the graphics state */
    public void init() {
        x = getWidth() / 2;
        y = getHeight() / 2;
        dx = SQUARE_SIZE;
        dy = 0;
        addMouseListeners();
    }
}
/* Runs the simulation */
public void run() {
    while (getElementAt(x, y) == null) {
        drawSquare();
        updateSnakeHeadLocation();
        pause(PAUSE_TIME);
    }
}

/* Adds a square to the window centered at (x, y) */
private void drawSquare() {
    GRect rect = new GRect(SQUARE_SIZE, SQUARE_SIZE);
    rect.setFilled(true);
    add(rect, x - SQUARE_SIZE / 2, y - SQUARE_SIZE / 2);
}

/* Updates the location of the snake head */
private void updateSnakeHeadLocation() {
    x += dx;
    y += dy;
}

/* Reacts to a mouse-pressed event */
public void mousePressed(MouseEvent e) {
    if (dy == 0) {
        dx = 0;
        dy = (e.getY() > y) ? SQUARE_SIZE : -SQUARE_SIZE;
    } else {
        dx = (e.getX() > x) ? SQUARE_SIZE : -SQUARE_SIZE;
        dy = 0;
    }
}

/* Instance variables */
private double x;      /* The x coordinate of the snake head */
private double y;      /* The y coordinate of the snake head */
private double dx;     /* The x velocity of the head */
private double dy;     /* The y velocity of the head */

/* Private constants */
private static final int PAUSE_TIME = 100;
private static final double SQUARE_SIZE = 15;
Problem 5: Using the String class (10 points)
Decomposition is not essential here. This solution introduces a second method just to provide a more general tool.

```java
/*
 * Returns a random key consisting of the 26 uppercase letters in
 * some randomly chosen order.
 */
private String generateRandomKey() {
    return generateRandomPermutation("ABCDEFGHIJKLMNOPQRSTUVWXYZ");
}

/*
 * Returns a random permutation of the letters passed in
 * as the argument str.
 */
private String generateRandomPermutation(String str) {
    String result = "";
    while (str.length() > 0) {
        int index = rgen.nextInt(str.length());
        result += str.charAt(index);
        String head = str.substring(0, index);
        String tail = str.substring(index + 1);
        str = head + tail;
    }
    return result;
}
```