Eric Roberts
CS 106A

Handout #46
May 14, 2012

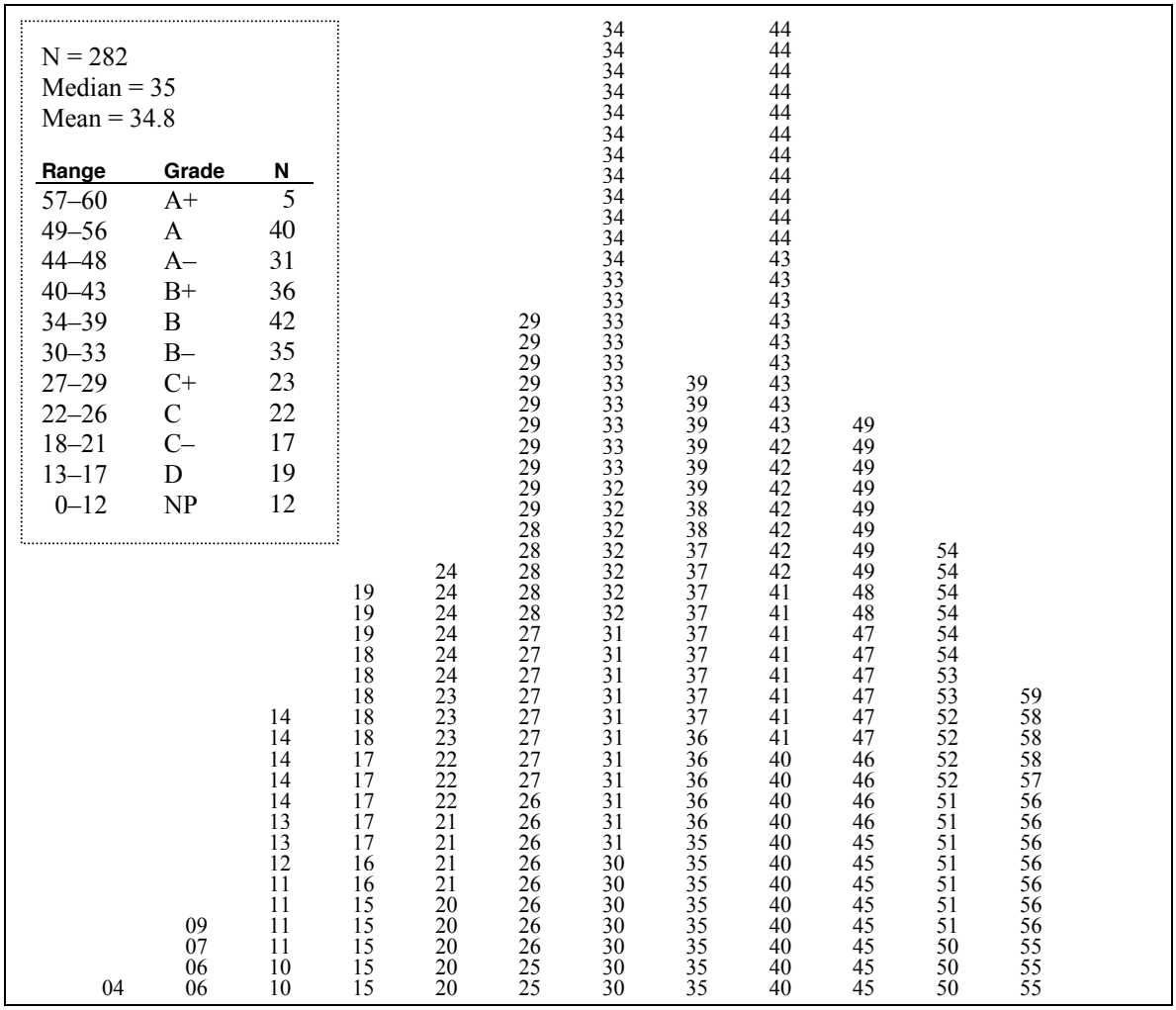
Answers to Midterm Exam

This midterm exam didn't go as well as midterms have in recent years. The median was a 58 percent, which is substantially lower than what I shoot for and lower than it has been in recent years. In retrospect, it's clear that the exam was too hard. Even so, it's also clear that many of you are struggling with developing algorithms for these types of problems.

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

N = 282
Median = 35
Mean = 34.8

Range	Grade	N
57–60	A+	5
49–56	A	40
44–48	A–	31
40–43	B+	36
34–39	B	42
30–33	B–	35
27–29	C+	23
22–26	C	22
18–21	C–	17
13–17	D	19
0–12	NP	12



Problem 1: Karel the Robot (10 points)

The key insight you need to make in solving this problem is that the simplest way to ensure that the beepers are evenly distributed is for Karel to put them down one at a time, cycling sequentially through the four directions. The only tricky part of this strategy is returning Karel to the center after each beeper placement. The sample solution accomplishes this goal by looking for the side streets. Other possibilities include painting this corner or filling in the center beeper only at the end.

```
/*
 * File: OccupyKarel.java
 * ----------------------
 * This program distributes beepers along the four streets leading out
 * of an intersection so that all the beepers are as close as possible
 * to the center of the action.
 */

import stanford.karel.*;

public class OccupyKarel extends SuperKarel {

   public void run() {
      while (beepersInBag()) {
         findFirstFreeCorner();
         putBeeper();
         turnAround();
         findInitialIntersection();
         turnLeft();
      }
   }

/*
 * Finds the first corner that does not contain a beeper.
 */
   private void findFirstFreeCorner() {
      while (beepersPresent()) {
         move();
      }
   }

/*
 * Finds the initial intersection, which is indicated by an opening to
 * the side street.
 */
   private void findInitialIntersection() {
      while (leftIsBlocked()) {
         move();
      }
   }

}
```

Problem 2: Simple Java programs (10 points)

(2a)

`(7 - 7 / 7 + 7) % 7 * 7 + 7 % 7`	42
`Character.isDigit((char) 2)`	`false`
`2 + 0 + "1" + '2'`	`"212"`

(2b)

```
"sunspots"
```

(2c)

```
x = 2, y = 1, z = 1
grumble grumble
x = 3, y = 1, z = 8
```

Problem 3: Simple Java programs (15 points)

The following solution uses arithmetic operators to isolate the digits; the problem is slightly easier to solve if you convert the integer `n` to a string first.

```
/*
 * Returns true if the decimal expansion of n contains the same
 * digit in three consecutive positions.
 */
  private boolean containsTriple(int n) {
     int count = 0;
     int prev = -1;
     while (n > 0) {
        int last = n % 10;
        if (last == prev) {
           count++;
           if (count == 3) return true;
        } else {
           count = 1;
           prev = last;
        }
        n /= 10;
     }
     return false;
  }
```

Problem 4: Using the graphics and random number libraries (15 points)

The sample solution below uses direction constants to indicate the direction of motion. Another approach (which actually results in a slightly shorter program) is to define **dx** and **dy** variables that keep track of how might **x** and **y** change in each time step.

```
/*
 * 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;

}
```

Problem 5: Strings and characters (10 points)

```
/*
 * Inverts a key for a letter-substitution cipher, where a key
 * is a 26-letter string that shows how each letter in the
 * alphabet is translated into the encrypted message.  For
 * example, if a key is "LZDRXPEAJYBQWFVIHCTGNOMKSU", that
 * means that 'A' (the first letter in the alphabet) translates
 * to 'L' (the first letter in the key), 'B' translates to 'Z',
 * 'C' translates to 'D', and so on.  The inverse of a key is
 * a 26-letter that translates in the opposite direction.  As
 * an example, the inverse of "LZDRXPEAJYBQWFVIHCTGNOMKSU" is
 * "HKRCGNTQPIXAWUVFLDYSZOMEJB"
 *
 */
   private String invertKey(String key) {
      String newKey = "";
      for (int i = 0; i < key.length(); i++) {
         char target = (char)('A' + i);
         int indexOfTarget = key.indexOf(target);
         newKey += (char)('A' + indexOfTarget);
      }
      return newKey;
   }
```