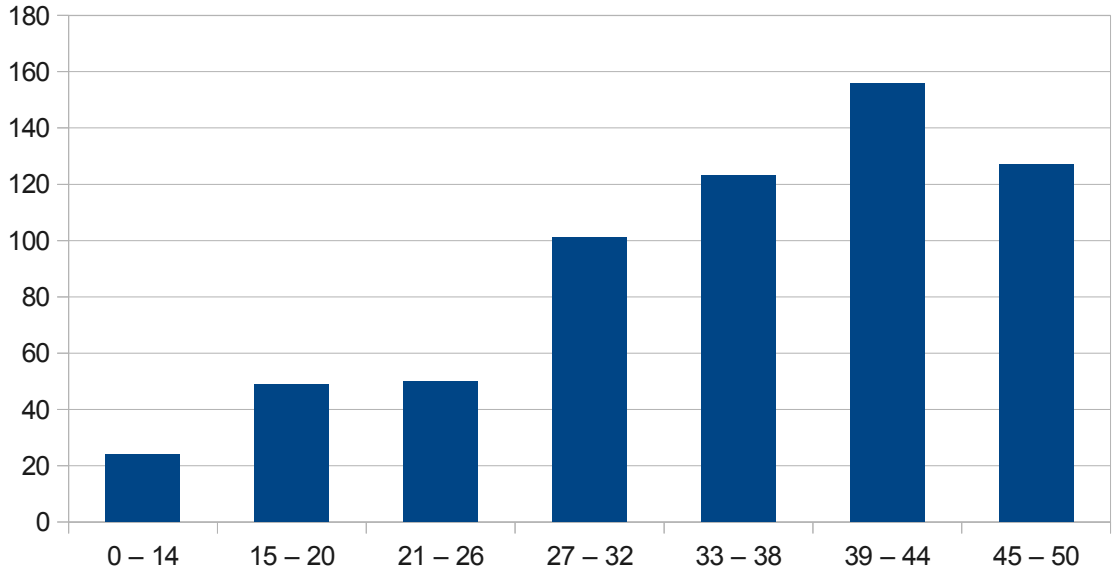
CS106A Midterm Exam Solutions

The grade distribution for this exam was as follows:



Overall, the final statistics were as follows:

75th Percentile: 43 / 50 (86%)
50th Percentile: 37 / 50 (74%)
25th Percentile: 29 / 50 (58%)

We are **not** grading this course using raw point totals and will instead be grading on a (fairly generous) curve. Roughly speaking, the median score corresponds to roughly a B+. As always, if you have any comments or questions about the midterm or your grade on the exam, please don't hesitate to drop by office hours with questions! You can also email Vikas or Keith with any questions.

If you think that we made any mistakes in our grading, please feel free to submit a regrade request to us. Just write a short (one-paragraph or so) description of what you think we graded incorrectly, staple it to the front of your exam, and hand your exam to either Vikas or Keith by Wednesday, February 26 at 3:15PM. We reserve the right to regrade your entire exam if you submit it for a regrade.

Problem 1: RectangleKarel (10 Points)

There are many solutions to this problem. The typical solution works by laying down one row/column at a time, backing up to the start of that row/column, then moving to the next row/column. The main challenges are not forgetting the first/last row/column, the first/last beeper in each row/column, and not crashing into a wall if Karel's initial position was flush up against a wall. Here's one possible solution:

```
import stanford.karel.*;
public class RectangleKarel extends SuperKarel {
    public void run() {
        while (frontIsClear()) {
            makeColumn();
            move();
        }
        makeColumn();
    }

    /* Fills all corners below or at Karel's position in the current column
     * with beepers, returning Karel back to the position Karel began in.
     */
    private void makeColumn() {
        fillDown();
        comeBackUp();
    }

    /* Fills the corners below or at Karel's position in the current column
     * with beepers. Karel begins facing East at some position and ends
     * facing South at the bottom of the column.
     */
    private void fillDown() {
        turnRight();
        putBeeper();
        while (frontIsClear()) {
            move();
            putBeeper();
        }
    }

    /* Moves Karel from the bottom of a row, facing South, back up to the position
     * of the first beeper on the row, facing East.
     */
    private void comeBackUp() {
        /* Ascend to the top of the world, then descend down to the beepers. */
        turnAround();
        while (frontIsClear()) {
             move();
        }
        turnAround();
        while (noBeepersPresent()) {
            move();
        }
        turnLeft();
    }
}
```

Problem Two: Jumbled Java hiJinks (10 Points Total)

(i) The Magic Number (6 Points)

For each of these programs, determine whether there are any values of x that can be entered so that the program will print **success** without causing any errors and without printing out anything else. If there are any values of x that will work, give any one of them. If there are no values of x that will work, write "no solution." No justification is necessary.

```
/* Program A */
int x = readInt();
if (x != 0 && x / 2 == 0) {
    println("success");
}
```

+1 and -1 are the only possible correct answers.

Answer for Program A: **1**

```
/* Program B */
int x = readInt();
if (x >= 10000) {
   while (x != 0) {
      if (x % 10 != 9) {
         println("failure");
      }
      x /= 10;
   }
   println("success");
}
```

Any positive integer made up only of 9's that has at least five 9's will work. Don't deduct points if they choose an integer that doesn't fit in an **int**; we never talked about that in this course.

Answer for Program B: **99999**

```
/* Program C */
int x = readInt();
if (x > 7 && x / 0 == x) {
    println("success");
}
```

If the operation doesn't short-circuit, it divides by zero and produces an error. If it does short-circuit, it evaluates to false, so success isn't printed.

Answer for Program C: **no solution**

```
/* Program D */
int x = readInt();
if (x != 0 || x != 1) {
    println("failure");
}
println("success");
```

Every number is either not 0 or not 1.

Answer for Program D: **no solution**

```
/* Program E */
int x = readInt();
if (1 - 3 - 5 - x == -10) {
    println("success");
}
```

3 is the only correct answer.

Answer for Program E: **3**

```
/* Program F */
int x = readInt();
if (x / 2 * 3 == 6) {
    println("success");
}
```

4 and 5 are correct answers.

Answer for Program F: **4**

(ii) Program Tracing **(4 Points)**

Determine the output of the following program and write it in the indicated box.

```
import acm.program.*;
public class FrozenJava extends ConsoleProgram {
	public void run() {
		int anna = 16;
		int elsa = 18;

		anna = arendelle(anna, elsa);
		println("anna = " + anna);
		println("elsa = " + elsa);
	}

	private int arendelle(int elsa, int anna) {
		String kristoff = "hans";

		weselton(kristoff);
		println("kristoff = " + kristoff);

		elsa = kristoff.length();
		return anna;
	}

	private void weselton(String olaf) {
		olaf += "el";
		println("olaf = " + olaf);
	}
}
```

Write the output of this program in the box below:

```
olaf = hansel
kristoff = hans
anna = 18
elsa = 18
```

Problem Three: Slicing a Cake (10 Points)

```
import acm.program.*;
import acm.util.*;

public class SlicingACake extends ConsoleProgram {
    private static final int NUM_TRIALS = 10;

    public void run() {
        int numPeople = readInt("How many people? ");
        int totalHappy = 0;
        for (int i = 0; i < NUM_TRIALS; i++) {
            int happyNow = sliceACake(numPeople);
            println("  Round " + (i + 1) + ": " + happyNow);
            totalHappy += happyNow;
        }
        double result = (double)totalHappy / NUM_TRIALS;
        println("Average happy people: " + result);
    }

    private int sliceACake(int numPeople) {
        double cakeLeft = numPeople;
        int result = 0;

        RandomGenerator rgen = RandomGenerator.getInstance();
        while (cakeLeft >= 2) {
            cakeLeft -= rgen.nextDouble(1.0, 2.0);
            result++;
        }
        if (cakeLeft >= 1) result++;
        return result;
    }
}
```

Problem Four: Pebbling a Checkerboard (10 Points)

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

public class PebblingACheckerboard extends GraphicsProgram {
    /* The size of a checker. */
    private static final double CHECKER_SIZE = 50;

    public void run() {
        addInitialCheckers();
        addMouseListeners();
    }

    /* Adds the initial three checkers to the world. */
    private void addInitialCheckers() {
        addChecker(0, getHeight() - CHECKER_SIZE);
        addChecker(0, getHeight() - 2 * CHECKER_SIZE);
        addChecker(CHECKER_SIZE, getHeight() - CHECKER_SIZE);
    }

    /* Adds a checker at the given position. */
    private void addChecker(double x, double y) {
        GOval checker = new GOval(x, y, CHECKER_SIZE, CHECKER_SIZE);
        checker.setFilled(true);
        add(checker);
    }

    /* Reacts to mouse clicks by adding checkers if appropriate. */
    public void mouseClicked(MouseEvent e) {
        GObject hit = getElementAt(e.getX(), e.getY());
        if (hit != null &&
            getElementAt(e.getX() + CHECKER_SIZE, e.getY()) != null &&
            getElementAt(e.getX(), e.getY() - CHECKER_SIZE) != null) {
            remove(hit);
            addChecker(hit.getX() + CHECKER_SIZE, hit.getY());
            addChecker(hit.getX(), hit.getY() - CHECKER_SIZE);
        }
    }
}
```

Problem Five: Identifying Plasmids (10 Points)

There are *many* possible solutions to this problem. Here are a few:

```
private boolean areSamePlasmids(String p1, String p2) {
    if (p1.isEmpty() && p2.isEmpty()) return true;
    if (p1.length() != p2.length()) return false;
    for (int i = 0; i < p1.length(); i++) {
       if (equalStartingAt(p1, p2, i)) return true;
    }
    return false;
}

private boolean equalStartingAt(String p1, String p2, int index) {
   for (int i = 0; i < p1.length(); i++) {
      if (p1.charAt(i) != p2.charAt((i + index) % p2.length())) {
         return false;
      }
   }
   return true;
}
```

```
private boolean areSamePlasmids(String p1, String p2) {
   if (p1.isEmpty() && p2.isEmpty()) return true;
   for (int i = 0; i < p1.length(); i++) {
      String rotated = p1.substring(i) + p1.string(0, i);
      if (rotated.equals(p2)) return true;
   }
   return false;
}
```

```
private boolean areSamePlasmids(String p1, String p2) {
   return p1.length() == p2.length() && (p1 + p1).indexOf(p2) != -1;
}
```