CIS 110 — Introduction to Computer Programming

13 February 2013 — Make-Up Midterm Midterm

Name: ______________________________

Recitation # (e.g. 201): ______________________________

Pennkey (e.g. bjbrown): ______________________________

My signature below certifies that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this examination.

______________________________ ______________________
Signature Date

Scores:

1		1
2		10
3		14
4		20
5		20
6		25
Total:		90

CIS 110 Exam Instructions

- You have 120 minutes to finish this exam. Time will begin when called by a proctor and end precisely 120 minutes after that time. If you continue writing after the time is called, you will receive a zero for the exam.
- **Make sure your phone is turned off before the exam starts. If it vibrates or rings during the exam, you will receive a substantial penalty.**
- Food and drink are strictly forbidden, **including water and gum**.
- You may not use your phone or open your bag for any reason, including to retrieve or put away pens or pencils, **until you have left the exam room**.
- This exam is *closed-book, closed-notes, and closed-computational devices.* Except where noted, code included in the questions is correct and you may use it as a reference for Java syntax.
- If you get stuck part way through a problem, it may be to your advantage to go on to another problem and come back later if you have time.
- All code must be written out as normal, including all curly braces and semicolons, unless the question states otherwise.
- Do not separate the pages of the exam. If a page becomes loose, write your name on it and use the provided staplers to reattach the sheet when you turn in your exam so that we don't lose it. We reserve the right not to grade any answers on loose sheets of paper.
- Turn in all scratch paper that you use during the exam. Do not take any sheets of paper with you or leave them behind.
- If you require extra paper, please use the backs of the exam pages or the extra sheet(s) of paper provided at the end of the exam. Clearly indicate on the question page where the graders can find the remainder of your work (e.g. "back of page" or "on extra sheet"). Staple an extra sheets you use to the back of your exam when you turn it in using the provided staplers.
- Use a pencil, or blue or black pen to complete the exam. All other colors are reserved for grading. If you do not have an appropriate writing utensil, raise your hand, and we will give you a pencil.
- If you have any questions, raise your hand and an exam proctor will come to answer them.
- When you turn in your exam, you may be required to show ID. If you forgot to bring your ID, talk to an exam proctor immediately.

Good luck and have fun!

Miscellaneous

1. (1 points)

   (a) Write your name, recitation number, and PennKey (username) on the front of the exam.

   (b) Sign the certification that you comply with the Penn Academic Integrity Code

Truth or Dare

2. (10 points) For each of the following boolean expressions, state whether it is always `true` (T), always `false` (F), sometimes `true` and sometimes `false`/not enough information to tell (S), will result in a compiler error (CE), or will result in a run-time exception (RE). Assume there are no rounding errors and the variables `x` and `y` are `int`s.

   (a) `2 * x / 2 == x` ____________

   (b) `2(3) == 6` ____________

   (c) `Math.sqrt(x) * Math.sqrt(x) == x` ____________

   (d) `Double.parseDouble(3.0) == 3.0` ____________

   (e) `Double.parseDouble("3") == 3.0` ____________

   (f) `1.0 + 2.0 >= 3.0` ____________

   (g) `x - Math.abs(x - y) != y` ____________

   (h) `1 / 0 != 1.0 / 0.0` ____________

   (i) `x - y <= x` ____________

   (j) `3 / 2 < 1.5` ____________

Bugs Bunny

3. (14 points) Identify 7 bugs in the program below that will prevent it from compiling or running. For each bug, give the line number and corrected line of code. Write your answers on the following page.

Since this program is loony anyway, we will accept any reasonable fix that allows the program to compile and run without error. You do not need to worry about the program's purpose.

```
 0  public class LittleBunny() {
 1      public static void main(String args) {
 2          int input = args[0];
 3          if (input < 1)
 4              return;
 5          double arr = new double[input];
 6          int i;
 7          for (int j = 0; j <= arr.length; j += 1) {
 8              i = (i + j) % arr.length;
 9              arr[j] = foofoo(i, j);
10              System.out.println("" + arr[j]);
11          }
12      }
13
14      public static String foofoo(int i, int j) {
15          if (i < j)
16              return "a";
17          if (i > j)
18              return "b";
19      }
20  }
```

Write your answers in the spaces on the following page.

Bugs Bunny (Cont'd)

Bug 1:

Bug 2:

Bug 3:

Bug 4:

Bug 5:

Bug 6:

Bug 7:

A Square Meal

4. (20 points) Each of the four figures below can be created by calling a recursive function `recursive(3, 0.5, 0.5, 0.25)` whose arguments are the recursive depth, x and y positions, and size. In each case, you can implement the function by reordering the six lines of code given below. Assume the `drawSquare()` function exists and draws a gray square with a black outline.

For each of the four figures, put the six lines of code in the correct order to generate the figure. You only need to give the numbers of each line; you do not need to rewrite them.

```
public static void recursive(int n, double x, double y, double sz) {
    1: recursive(n - 1, x - sz, y + sz, sz / 2) // upper left
    2: recursive(n - 1, x + sz, y + sz, sz / 2) // upper right
    3: recursive(n - 1, x + sz, y - sz, sz / 2) // lower right
    4: recursive(n - 1, x - sz, y - sz, sz / 2) // lower left
    5: drawSquare(x, y, sz)
    6: if (n == 0) return;
}
```

Line 1: ____________

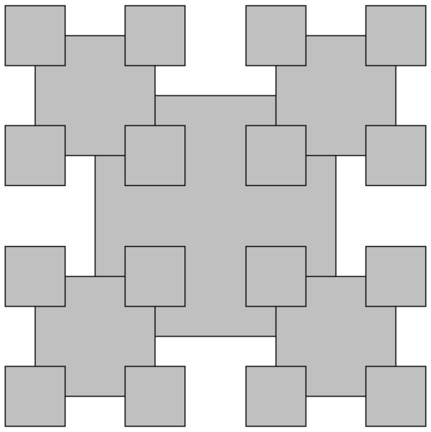
Line 2: ____________

Line 3: ____________

Line 4: ____________

Line 5: ____________

Line 6: ____________



Line 1: ____________

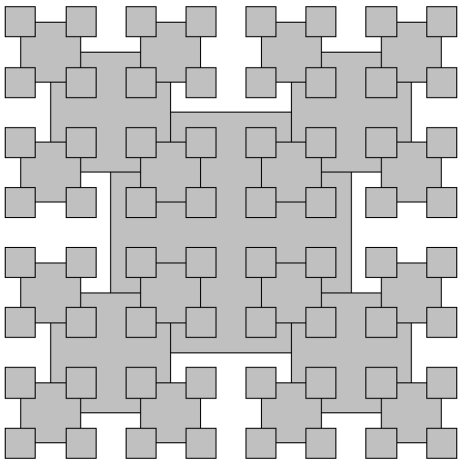
Line 2: ____________

Line 3: ____________

Line 4: ____________

Line 5: ____________

Line 6: ____________



A Square Meal (Cont'd)

Rearrange these six lines of code so they draw each of the following figures:

```
public static void recursive(int n, double x, double y, double sz) {
    1: recursive(n - 1, x - sz, y + sz, sz / 2) // upper left
    2: recursive(n - 1, x + sz, y + sz, sz / 2) // upper right
    3: recursive(n - 1, x + sz, y - sz, sz / 2) // lower right
    4: recursive(n - 1, x - sz, y - sz, sz / 2) // lower left
    5: drawSquare(x, y, sz)
    6: if (n == 0) return;
}
```

Line 1: ____________

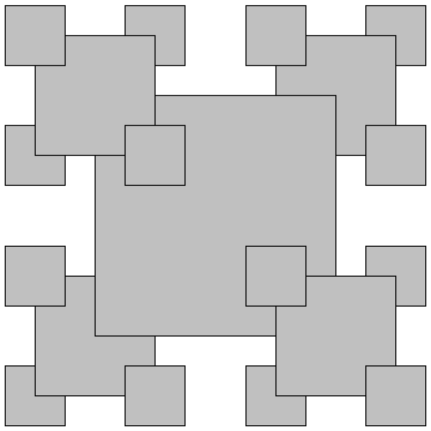
Line 2: ____________

Line 3: ____________

Line 4: ____________

Line 5: ____________

Line 6: ____________



Line 1: ____________

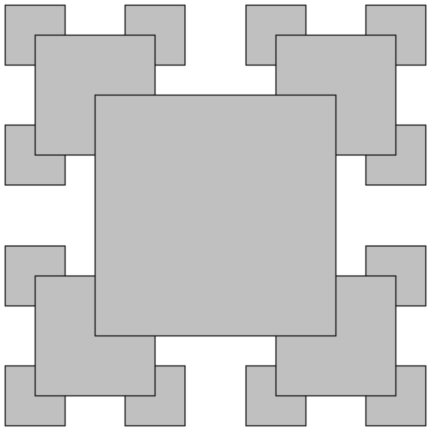
Line 2: ____________

Line 3: ____________

Line 4: ____________

Line 5: ____________

Line 6: ____________



Recess

5. (20 points) Read the code below, then answer the questions on the next page:

```
public class Playground {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);

        if (n == 1) {
            int d = slide(Integer.parseInt(args[1]),
                          Integer.parseInt(args[2]),
                          Integer.parseInt(args[3]));

            System.out.println((d + "-") + (d + 5 + "-") +
                               ((d + 12) + "-") + d + ("-" + 83));
        } else if (n == 2) {
            double d = monkeybars(Integer.parseInt(args[1]));
            System.out.println(d);
        } else if (n == 3) {
            String d = tetherball();
            System.out.println(d);
        }
    }

    public static int slide(int a, int b, int c) {
        if (b < 1) return (a + c);
        a *= 2;
        return slide(a, b-1, c);
    }

    public static int monkeybars(int a) {
        while (a < 81) a += 2;
        return a;
    }

    public static String tetherball() {
        String[] arr = {"M", "U", "2", "R", "Y"};
        return arr[4] + arr[1] + arr[Integer.parseInt(arr[2]) -
                                     Integer.parseInt(arr[2])];
    }
}
```

Recess (Cont'd)

(a) What does the command "`java Playground 1 2 5 3`" print? Circle your answer.

(b) What does the command "`java Playground 2 2`" print? Circle your answer.

(c) What does the command "`java Playground 3`" print? Circle your answer.

(d) Describe **in 20 words or less** what `slide()` computes. You may assume that `a`, `b`, and `c` are all at least zero. Circle your answer.

(e) Describe **in 20 words or less** what `monkeybars()` doess. You may assume that `a` is at least zero. Circle your answer.

What a Dupe

6. (25 points) This question consists of three parts **on three pages**. For each part, you only need to write the prescribed function; you do not need to write the surrounding class. You also do not need to write any comments.

(a) Write a function `contains()` that takes an integer `x` and an array of integers `arr`, and returns `true` or `false` depending on whether or not `arr` contains the value `x`. You may assume that `arr` contains at least one element.

What a Dupe (Cont'd)

(b) Write a function `dupes()` that accepts two arrays of integers and uses the `contains()` function to compute and return the number of values that occur in both arrays. Your solution must not be recursive. You may assume that each array contains at least one value, that no value occurs more than once within either array, and that the `contains()` function is defined in the same class as `dupes()`.

What a Dupe (Cont'd)

(c) Write a recursive function, `dupes2()` that accepts two arrays of integers and an integer `n`. `dupes2(arr1, arr2, n)` should do the same thing as `dupes()` — use the `contains()` function to compute and return the number of values that occur in both `arr1` and `arr2` — except that it should only consider values in entries `n` *and higher* of `arr1`. `dupes2(arr1, arr2, 0)` should therefore return the same result as `dupes(arr1, arr2)`.

`dupes2()` must not contain any loops and must not call `dupes()`. You may assume that each array contains at least one value, that no value occurs more than once within either array, and that the `contains()` function is defined in the same class as `dupes2()`. You may also assume that `dupes2()` will only be called with a value of `n` that is at least 0.

Postscript (extra paper)

Postscript (extra paper)