CIS 110 Spring 2013 Midterm, 12 February 2013, Answer Key

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) `Double.parseDouble(3.0) == 3.0` CE

   (b) `3 / 2 == 1.5` F

   (c) `2 * x / 2 == x` T

   (d) `2(3) == 6` CE

   (e) `1 / 0 == 1.0 / 0.0` RE

   (f) `Math.sqrt(x) * Math.sqrt(x) == x` S

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

   (h) `1.0 + 2.0 == 3.0` T

   (i) `Double.parseDouble("3") == 3.0` T

   (j) `x - y <= x` S

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  }
```

Bug 1: 0: `public class LittleBunny {`
Bug 2: 1: `public static void main(String[] args)`
Bug 3: 2: `int input = Integer.parseInt(args[0])`
Bug 4: 5: `String[] arr = new String[input]`
Bug 5: 6: `int i = 0;`
Bug 6: 7: `for (int j = 0; j < arr.length; j += 1)`
Bug 7: 17: `else` -or- `if (i >= j)` -or- insert add return statement at line 19 -or- remove line 17.

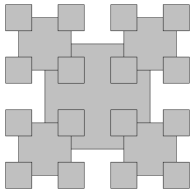
A Square Meal

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.

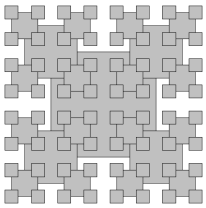
For each of the four figures, put the six lines of code in the correct order to generate the figure. You only need to give the numbers of each line; you do not need to rewrite them.

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

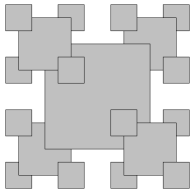
Line 1: 4
Line 2: 1
Line 3: 2
Line 4: 3
Line 5: 5
Line 6: 6



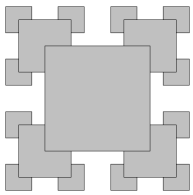
Line 1: 1
Line 2: 4
Line 3: 2
Line 4: 3
Line 5: 5
Line 6: 6



Line 1: 4
Line 2: 2
Line 3: 6
Line 4: 1
Line 5: 3
Line 6: 5



Line 1: 4
Line 2: 2
Line 3: 3
Line 4: 5
Line 5: 6
Line 6: 1



Recess

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

(a) What does the command "`java Playground 1 2 5 3`" print? Circle your answer.
`67-72-79-67-83`
(The ASCII character codes for CHOCS)

(b) What does the command "`java Playground 2 2`" print? Circle your answer.
`82.0`
(The ASCII code for R)

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

(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.
Return $\texttt{a} * 2^{\texttt{b}} + \texttt{c}$.

(e) Describe **in 20 words or less** what `monkeybars()` doess. You may assume that `a` is at least zero. Circle your answer.
Increases `a` by 2 until exceeds 80, then returns it.

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.

```
public static boolean contains(int x, int[] arr) {
    for (int i = 0; i < arr.length; i++)
        if (arr[i] == x) return true;

    return false;
}
```

(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()`.

```
public static int dupes(int[] arr1, int[] arr2) {
    int d = 0;
    for (int i = 0; i < arr1.length; i++)
        if (contains(arr1[i], arr2)) d++;
    return 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.

```
public static int dupes2(int[] arr1, int[] arr2, int n) {
    if (n >= arr1.length) return 0;

    if (contains(arr1[n], arr2)) return 1 + dupes2(arr1, arr2, n + 1);
    else return dupes2(arr1, arr2, n + 1);
}
```