CIS 110 — Introduction to Computer Programming

12 February 2013 — 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:

0		1
1		9
2		10
3		15
4		20
5		30
Total:		85

CIS 110 Exam Instructions

- You have 115 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

0. (1 point)

(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

Types and Values

1. (9 points) Give the type and value for each of the following expressions and variable declarations. When there is more than one statement, give the type and value that the *last* statement computes. If the code would result in a syntax error, write "N/A" in the first column, and give the reason for the error in the second column. You do not need to write the exact error message.

	Type	Value
`6 / 4`		
`String s = "Hello"; s.length;`		
`0.0 / 0`		
`"Hello " + 4 + 3`		
`!!!false && !!true`		
`String s = StdOut.print("Hello");`		
`Math.pow(5, 2)`		
`6++`		
`char[] arr = {'5', '3'}; arr[2];`		

Find the Bugs

2. (10 points) The following program is supposed to print out the second-largest double value specified on the command line, or `-Infinity` if less than two numbers are given. But Hanssen didn't test it before sending it to Dr. Brown, and it's full of bugs. Write the line number and **corrected line of code** for ten bugs in the program. Assume that all arguments represent valid `double` values and that all the values are different.

Write your answers on the following page.

```
 1 public static void class SecondBiggest {
 2     public static void main(String[] args) {
 3       int N = args.length;
 4       int[] input = new int[N];
 5       for (int i = 1; i <= N; i++)
 6           input[i] = args[i];
 7       System.out.println(second(input));
 8     }
 9
10     public static double second(double[] arr){
11         double biggest = Double.NEGATIVE_INFINITY;
12         double secondBiggest = Double.POSITIVE_INFINITY;
13         if (arr.length  < 2) return;
14         for (int i = 0, i < arr.length, i++) {
15             double temp = arr[i];
16             if (arr[i] > biggest) biggest = temp;
17         }
18
19         for (int j = 0; j < arr.length; j++) {
20             temp = arr[j];
21             if ((temp > secondBiggest) && (temp != biggest)) {
22                 secondBiggest = temp;
23         }
24         return secondBiggest
25     }
26 }
```

Find the Bugs (Cont'd)

Bug 1:

Bug 2:

Bug 3:

Bug 4:

Bug 5:

Bug 6:

Bug 7:

Bug 8:

Bug 9:

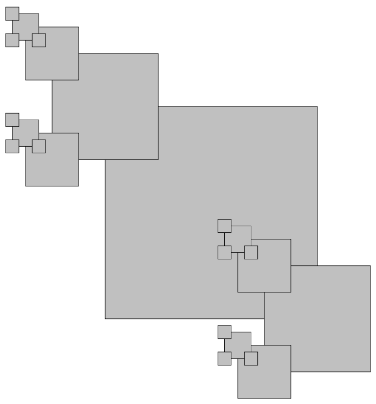
Bug 10:

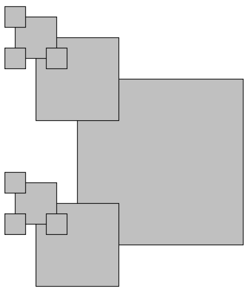
Recursive Graphics

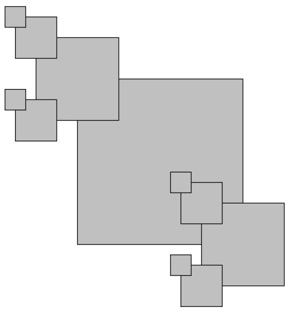
3. (15 points) The figures in this question were all generated using the function below. Assume the `graySquare()` function draws a gray square with a black outline. For each figure, give values of `r` and `z` that could draw it. Assume the starting values of `x` and `y` are 0.5, and that the starting value of `z` is an integer between 0 and 10.

Hint: Don't worry about scale. We will accept *any* values of `r` and `z` that could generate the figures below at some scale. However, the values we used were all multiples of one eighth.

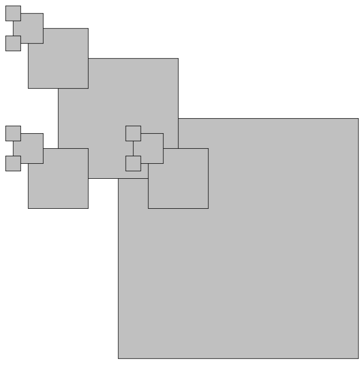
```
    public static void draw(double x, double y, double r, double z){
      graySquare(x, y, r);

      if (r < 0.05) return;

      if (z % 2 == 0) draw(x - r, y - r, r / 2, z - 1); // lower left
      if (z % 1 == 0) draw(x - r, y + r, r / 2, z - 1); // upper left
      else if (r < 1) draw(x + r, y + r, r / 2, z - 1); // upper right
      if (z % 3 == 0) draw(x + r, y - r, r / 2, z - 1); // lower right
    }
  }
```

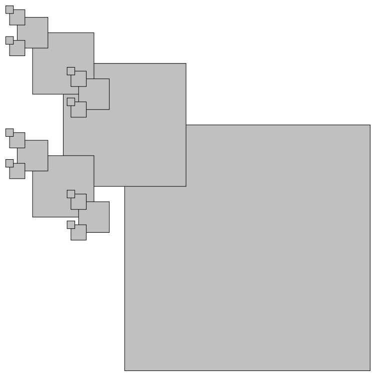


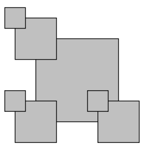




`r:` `z:` `r:` `z:` `r:` `z:`







`r:` `z:` `r:` `z:` `r:` `z:`

Prime Directive

4. (20 points) Using the Allen Telescope Array, the SETI ("Search for Extra-Terrestrial Intelligence") Institute recently picked up an unusual radio signal originating in the Orion nebula. Detailed, top-secret analysis of the signal has revealed it to be a Java program, giving indisputable proof of the language's universal appeal. Unfortunately, nobody has been able to figure out yet what the program does:

```
public class Vulcan {
  public static void main(String[] romulan) {
    int klingon = Integer.parseInt(romulan[0]);
    if (klingon < 1) return;

    for (int ferengi = 2; ferengi <= klingon; ferengi++) {
      if (klingon % ferengi == 0) {
        while (klingon % ferengi == 0) klingon /= ferengi;
        System.out.println(ferengi);
      }
    }
  }
}
```

(a) What does the program `Vulcan` print out when run with the argument "0"?

(b) What does `Vulcan` print out when run with the argument "1"?

(c) What does `Vulcan` print out when run with the argument "3"?

(d) What does `Vulcan` print out when run with the argument "8"?

(e) What does `Vulcan` print out when run with the argument "24"?

(f) What does `Vulcan` do? Your answer should be at most 30 words.

The Sum of its Parts

5. (30 points) Write two versions of a static function, `sumDigits`, that takes an integer and returns the sum of its digits. For example, the sum of the digits in 1234 is 10 because $1+2+3+4 = 10$, and the sum of the digits in -52 is 7 (the minus sign is not a digit, after all).

The first version, **on this page**, should be iterative and must not make any function calls (not even to library functions like `Math.max()`). The second version, **on the next page**, should be recursive but must not call any other functions and must not contain any loops. You do not need to include comments in either version, and you do not need to write a class around them.

Hint: Start with the version of the function that is most intuitive to you.

(a) Iterative version (no function calls):

The Sum of its Parts (cont'd)

(b) Recursive version (no loops):