CIS 110 Summer 2012 Midterm, 7 June 2012, 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

Find the Bugs

2. (5 points) Identify five errors in the following sorting routine that prevent it from compiling. The line numbers are included for your convenience and are not part of the program.

```
 1:  public static int selectionSort(int[] input) {
 2:    for (int i = 0; i < input.length; i++)
 3:      int minIndex = i;
 4:      for (int j = i; j < input.length; j++) {
 5:        if (input[j] < input[minIndex])
 6:          int minIndex = j;
 7:      }
 8:
 9:      int temp = input[i];
10:      input[i] = input[minIndex]
11:      input[minIndex] = temp;
12:    }
13:
14:    return input[];
15:  }
```

Bug 1: 1: should return `int[]`
Bug 2: 2: missing curly brace
Bug 3: 6: don't redeclare `myIndex`
Bug 4: 10: missing semi-colon
Bug 5: 14: no `[]` after `input`

Types and Expressions

3. (6 points) Give the type and value of each of the following Java expressions. If an expression will crash or will not compile, write `Illegal` under type and put an X in value. You must fill in every entry. Entries left blank will be marked incorrect.

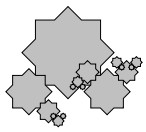
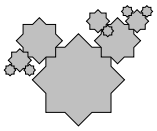
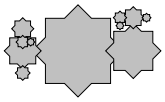
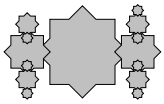
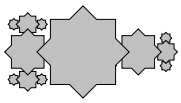
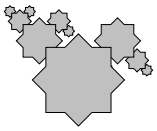
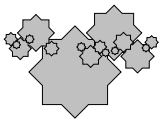
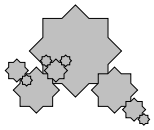
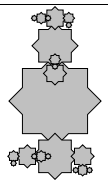
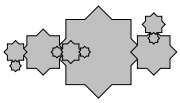
Expression	Type	Value
`3 < 4 < 5`	Illegal	X
`5 / 2`	int	2
`"5" + Math.max(3, 4)`	String	`"54"`
`(double) 2 / 4`	`double`	`0.5`
`(!!true) && (!!(!false))`	`boolean`	`true`
`Integer.parseInt(2)`	Illegal	X

Recursive Graphics

4. (13 points) Consider the following recursive function, then answer the questions on the following page. Assume that the helper function `drawShadedStar()` draws an eight-pointed, shaded gray star of radius `r` that is outlined in black and centered on `(x, y)`.

```
public static void starryNight(double x, double y, double r, boolean odd) {
  // Select a random integer from the set [0, 1, 2, 3]
  int randomInt = (int) (4 * Math.random());

  double new_r  = 0.5 * r;
  double offset = 0.6 * r;

  if (r < 0.02) return;

  drawShadedStar(x, y, r);

  if (randomInt % 2 == 0)
    starryNight(x - offset, y - offset, new_r, !odd);
  else if (randomInt / 2 == 0)
    starryNight(x + offset, y + offset, new_r, !odd);
  if (randomInt / 2 == 1)
    starryNight(x + offset, y - offset, new_r, !odd);
  else if (randomInt % 2 == 1)
    starryNight(x - offset, y + offset, new_r, !odd);
}
```

Assume a client issues the call `starryNight(0.5, 0.5, 0.25, true)`. For each figure below, state that it is **correct**, or indicate what feature(s) of the figure make it incorrect. Each inccorect figure has at most two problems.

	recursion too deep		correct
	rotated, star sizes are not consistent		rotated
	rotated, big stars in front of small ones		big stars in front of small ones
	star sizes are not consistent		correct
	rotated, recursion too deep		rotated

Tracery

5. (15 points)

For each of the labeled points in the code fragment below, identify each of the assertions in the table as being *sometimes*, *always*, or *never* true. Assume that `bar` is only called from within `foo`, and that the values of all `int`s stay within the valid range for integers (i.e. no value will grow so large that it will wrap around become negative, or vice cersa).

Abreviate sometimes with **S**, always with **A**, and never with **N**.

```
public static int foo(int x, int y, int z) {
  x = x * x;
  if (x < 0) y = y * y;
  else y = y - 2 * y;
  // POINT A

  if (x < 0) z = y;
  else y++;
  // POINT B

  if (z < 0) z = x;

  if (x > y * y) {
    // POINT C
    return bar(y / 3, x + 1, z - 1);
  } else {
    // POINT D
    return bar(y + 1, x, z - 1);
  }
}

public static int bar(int x, int y, int z) {
  if (z == 0) return 42;

  x = x * 4;
  // POINT E
  return foo(x / 2, -y - Math.abs(x), z - 1);
}
```

	`x >= 0`	`y > x`	`z < 0`
A	A	S	S
B	A	S	S
C	A	N	N
D	A	S	N
E	S	S	S

Partial Sums

6. (20 points) Write a function `sumUpTo` that takes a single argument `N`, reads in `N` integers from standard input using `StdIn.readInt()`, and returns an integer array of length `N` where the value at index `i` is the sum of the last `i + 1` numbers read. For example, if you read in the values 1, 3, -1, 4, 2, you should produce and return an array with the values 2, $2 + 4$, $2 + 4 + (-1)$, $2 + 4 + (-1) + 3$, $2 + 4 + (-1) + 3 + 1$, i.e. {`2, 6, 5, 8, 9`} . Any implementation that works will receive a good grade, but for full credit your function should create only one array. You may assume that `N` is at least 1 and that `StdIn.readInt()` always succeeds.

```
public static int[] sumUpTo(int N) {
    int[] arr = new int[N];

    for (int i = 0; i < N; i++)
        arr[N - i - 1] = StdIn.readInt();
    for (int i = 1; i < N; i++)
        arr[i] = arr[i] + arr[i - 1];

    return arr;
}
```