COS 126 **Princeton University** **Spring 2024**

Written Exam 1

This exam has 10 questions worth a total of 100 points. You have 80 minutes.

Instructions. This exam is preprocessed by computer. Write neatly, legibly, and darkly. Put all answers (and nothing else) inside the designated spaces. *Fill in* bubbles and checkboxes completely: ● and ■. To change an answer, erase it completely and redo.

Resources. The exam is closed book, except that you are allowed to use a one page reference sheet (8.5-by-11 paper, one side, in your own handwriting). No electronic devices are permitted.

Honor Code. This exam is governed by Princeton's Honor Code. Discussing the contents of this exam before solutions are posted is a violation of the Honor Code.

Please complete the following information now.

Name:

NetID:

Exam room: ○ McCosh 10 ○ McCosh 50 ○ McCosh 66 ○ Other

Precept:

P01	P02	P03	P03A	P03B	P04	P04A	P05	P05A	P06
○	○	○	○	○	○	○	○	○	○
P10	P10A	P10B	P11	P12	P13	P14	P14A	P15	P15A
○	○	○	○	○	○	○	○	○	○

"I pledge my honor that I will not violate the Honor Code during this examination."

Signature

1. **Initialization. (2 points)**

   In the designated spaces on the front of this exam,

   - *Write* your name.

   - *Write* your Princeton NetID (6–8 alphanumeric characters).

   - *Fill in* the bubble corresponding to where you are taking this exam.

   - *Fill in* the bubble corresponding to your precept.

   - *Write* and *sign* the Honor Code pledge.

2. **Java expressions. (12 points)**

Assume that the variables `x`, `y`, and `z` have been declared and initialized as follows:

```
int x = 1;
int y = 2;
int z = 3;
```

For each Java expression on the left, write the letter of the best-matching value from the right. You may use each letter once, more than once, or not at all.

- [H] `1 + 1`
- □ `y - x - 1`
- □ `z / (y - 2)`
- □ `(double) z / (y - 2)`
- □ `x * x + z * z / y * y`
- □ `Math.sqrt(x + z + 5) / y`
- □ `!!!(x == y == z)`
- □ `Integer.parseInt(x + "" + y + "" + z)`
- □ `(y / z) < (z % x)`

A. `false`

B. `true`

C. `0`

D. `1`

E. `1.0`

F. `1.5`

G. `1.75`

H. `2`

I. `2.5`

J. `6`

K. `9`

L. `10`

M. `123`

N. *positive infinity*

O. *compile-time error or run-time exception*

3. **Programming terminology. (10 points)**

For each programming term on the left, write the letter of the best-matching description from the right. Use each letter exactly once.

□ API

□ Array

□ Command-line arguments

□ Compile-time error

□ Data type

□ Expression

□ Literal

□ Run-time exception

□ Standard input

□ Variable

A. An error that indicates an invalid Java program.

B. An error that arises while the programming is executing.

C. A storage location for a data-type value.

D. Source code representation of a data-type value.

E. A combination of variable names, literals, operators, and function calls that evaluates to a value.

F. An indexed sequence of values of the same type.

G. Specifies method headers and behavior of a library.

H. Data provided to a program *before* it begins execution.

I. Data that a program receives *during* execution.

J. A set of values and operations on those values.

4. **Arrays, loops, and conditionals. (12 points)**

Let a[] be an array of type int[] and let n be its length. Determine what each of the following code fragments does. Assume that n is a positive integer and that all of the integers in the array a[] are positive integers.

For each code fragment on the left, write the letter of the best-matching description from the right. You may use each letter once, more than once, or not at all.

□

```
int result = 0;
for (int i = 0; i < n; i++) {
    if (a[i] > result)
        result = a[i];
}
```

□

```
boolean result = true;
for (int i = n-1; i >= 1; i--) {
    if (a[i] > a[i-1])
        result = false;
}
```

□

```
for (int i = 0; i < n; i++) {
    int temp = a[i];
    a[i] = a[n-i];
    a[n-i] = temp;
}
```

□

```
for (int i = 0; i < n/2; i++) {
    int temp = a[n-i-1];
    a[n-i-1] = a[i];
    a[i] = temp;
}
```

A. *Reverses* the elements in the array

B. *Sorts* the elements in the array

C. Identifies the *smallest* value in the array

D. Identifies the *largest* value in the array

E. Determines if the array is in *ascending* order

F. Determines if the array is in *descending* order

G. Produces a *compile-time error* or a *run-time exception*

H. Goes into an *infinite loop*

5. **Properties of functions. (12 points)**

   Which of the following are properties of *functions* in Java?

   Identify each statement as either true or false by filling in the appropriate bubble.

   true *false*

   ● ○ In Java, a *function* is implemented as a `static` method.

   ○ ○ A non-`void` function must contain *exactly one* `return` statement.

   ○ ○ A function can *both* return a value *and* produce a side effect.

   ○ ○ Two functions defined in the same class can have *both* the same name *and* the same number of arguments.

   ○ ○ A function can specify `boolean[]` as its *return type.*

   ○ ○ If you pass a value of type `double[]` to a function that takes an argument of type `double[]`, that function can change the values of the individual elements in the array.

   ○ ○ If you pass a value of type `int` to a function that takes an argument of type `double`, that will produce a *compile-time error.*

6. **Conditionals, loops, and standard drawing. (12 points)**

Consider the following code fragment, which draws an n-by-n grid of filled circles. Recall that `StdDraw.filledCircle(x, y, r)` draws a filled circle of radius r, centered at (x, y), in the current pen color.

```
// lower-left endpoint = (0, 0); upper-right endpoint = (n, n)
StdDraw.setXscale(0, n);
StdDraw.setYscale(0, n);

// draw an n-by-n grid of filled circles
for (int x = n-1; x >= 0; x--) {   // line 6
   for (int y = 0; y < n; y++) {   // line 7
      StdDraw.setPenColor(StdDraw.BLACK);
      if ((x == y) || (x + y == n-1)) StdDraw.setPenColor(StdDraw.RED);
      else {
         if (x % 2 == 0) StdDraw.setPenColor(StdDraw.GREEN);
         if (y % 2 == 0) StdDraw.setPenColor(StdDraw.BLUE);
      }
      StdDraw.filledCircle(x + 0.5, y + 0.5, 0.5);
   }
}
```

Which of the following properties are true for $n = 100$?

Fill in all checkboxes that apply.

■ It draws an n-by-n grid of filled, non-overlapping, circles.

□ The lower-leftmost and upper-rightmost circles are both red.

□ The upper-leftmost and lower-rightmost circles are the same color.

□ All circles are colored red, green, or blue (i.e., not black).

□ The number of green circles equals the number of blue circles.

□ The upper-leftmost circle is drawn *last.*

□ If lines 6 and 7 are swapped, the code fragment produces exactly the same final drawing (but the circles are drawn in a different order).

7. **Functions, arrays, and pass-by-value. (10 points)**

Consider the following Java functions:

```
public static int negate1(int x) {
    x = -x;
    return x;
}

public static void negate2(int[] a) {
    for (int i = 0; i < a.length; i++)
        negate1(a[i]);
}

public static void negate3(int[] a) {
    for (int i = 0; i < a.length; i++)
        a[i] = -a[i];
}

public static void negate4(int[] a) {
    for (int i = 0; i < a.length; i++)
        a[i] = negate1(a[i]);
}
```

Suppose that the integer array `a[]` contains the three integers `[1, 2, 6]`. What will be the contents of the array `a[]` after executing each of the following statements?

For each statement on the left, write the letter of the best-matching description from the right. Answer the parts independently. You may use each letter once, more than once, or not at all.

□ `a = -a;`

□ `negate2(a);`

□ `negate3(a);`

□ `negate4(a);`

□ `negate3(negate3(a));`

A. `[1, 2, 6]`

B. `[-1, -2, -6]`

C. `[0, 0, 0]`

D. Produces a *compile-time error* or a *run-time exception*

8. **Boolean expressions and functions. (10 points)**

The `exactly2()` function takes three integer arguments and returns `true` if exactly two of the three integers are equal; otherwise, it returns `false`. For example `exactly2(2, 2, 6)` returns `true`, but `exactly2(1, 2, 6)` and `exactly2(3, 3, 3)` return `false`.

Complete the following *two* implementations of `exactly2()` by, for each oval numbered 1–5, choosing one of the boolean expressions from the right. No other code is allowed.

Each answer should be a sequence of uppercase letters (corresponding to the labeled ovals). You may use each letter once, more than once, or not at all.

A. `a == b`

B. `a == c`

C. `b == c`

D. `a != b`

E. `a != c`

F. `(a == c) || (b == c)`

G. `(a == c) && (b == c)`

(a)

1	2	3

```
public static boolean exactly2(int a, int b, int c) {

    if ( 1 ) return false;

    else if ( 2 ) return true;

    else if ( 3 ) return true;

    return false;

}
```

(b)

4	5

```
public static boolean exactly2(int a, int b, int c) {

    int count = 0;

    if ( 4 ) count++;

    if ( 5 ) count++;

    return count == 1;

}
```

9. **Recursion. (10 points)**

Consider the following *recursive* Java function:

```
public static int f(int x) {
    if (x < 0) return 0;
    else if (x < 5) return x;
    int sum = f(x-1) + f(x-2) * f(x-3);
    return sum % 5;
}
```

Fill in the bubble corresponding to the value of each expression below.

(a) `f(-126)`

0 1 2 3 4

○ ○ ○ ○ ○

(b) `f(4)`

0 1 2 3 4

○ ○ ○ ○ ○

(c) `f(6)`

0 1 2 3 4

○ ○ ○ ○ ○

(d) `f(10)`

0 1 2 3 4

○ ○ ○ ○ ○

(e) `f(126)`

0 1 2 3 4

○ ○ ○ ○ ○

10. **Performance. (10 points)**

Determine the *order-of-growth of the running time* of each of the following code fragments as a function of n.

For each code fragment on the left, write the letter of the best-matching term from the right. You may use each letter once, more than once, or not at all.

□

```
int count = 0;
```

□

```
int count = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j++) {
        count++;
    }
}
```

□

```
int count = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j = 2*j) {
        count++;
    }
}
```

□

```
public static int f(int n) {
    if (n == 0) return 1;
    return f(n-1) + f(n-1);
}
```

□

```
int count = 0;
for (int i = 1; i <= n; i++)
    count++;
for (int j = 1; j <= 2*n; j++)
    count++;
for (int k = 1; k <= 3*n; k++)
    count++;
```

A. $\Theta(1)$
constant

B. $\Theta(\log n)$
logarithmic

C. $\Theta(n)$
linear

D. $\Theta(n \log n)$
linearithmic

E. $\Theta(n^2)$
quadratic

F. $\Theta(n^3)$
cubic

G. $\Theta(2^n)$
exponential