Name:

Login ID:

Precept:

COS 126 Midterm 1 Written Exam Fall 2012

This test has 8 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a one page single-sided cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Print your name, login ID, and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. *Note*: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in the class has taken the exam. You have 50 minutes to complete the test.

"I pledge my honor that I have not violated the Honor Code during this examination."

____

Signature

1	/7
2	/9
3	/12
4	/6
5	/8
6	/6
7	/12
8	/10
TOTAL	/70

1. Types and Casts (7 points). Give the value and type of each of the following Java expressions. For any expression that will not compile, write `X` for its value and `Illegal` for its type.

expression	*value*	*type*
`Integer.parseInt(“123”)`	123	int
`1 + 23 + "45"`	“2445” [int addition then String concatenation]	String
`!((!true || true) && false)`	true	boolean
`1 = 2`	X	Illegal [Cannot assign a value to 1]
`(1 < 2) == false`	false	boolean
`3 * 0.5 + 3 / 2 * 0.5`	2.0 [1.5 + (1 * 0.5)]	double
`(double)1/4`	0.25	double

2. Miscellaneous (9 points).

A. Which of the following are true of Java arrays? Circle *all* that apply.

(i) Array entries are auto-initialized to `0.0` when creating a new array of `double` values.

(ii) ~~Can change the size of the array after creation.~~

(iii) Given an array `a[]` that has been declared and initialized, accessing `a[a.length]` results in a runtime error.

(iv) Can use an array as a return type from a function.

(v) Can pass an array to a function and have that function change the values stored in the array entries.

B. Which of the following best defines a *data type*? Circle the *single best* answer.

(i) ~~A set of values.~~

(ii) ~~A set of operations.~~

(iii) ~~A sequence of 0s and 1s.~~

(iv) A set of values and operations on those values.

(v) ~~The type of the arguments, method name, and the type of the return value for a function.~~

C. Why might a program be written to take input from standard input rather than to take command-line arguments? Circle the *single best* answer.

(i) ~~To allow the usage of inputs that consist of more than one line.~~

(ii) ~~To allow the usage of inputs that come from a file.~~

(iii) ~~To allow the program to interpret inputs adaptively according to prior inputs.~~

(iv) All of the above.

(v) ~~None of the above.~~

3. Arrays (12 points). Consider the following code.

```
public class ArrayQ
{
   public static void main(String[] args)
   {
      int N = 6;
      int[] a = new int[N];
      for (int i = 0; i < N; i++)
         a[i] = StdIn.readInt();

      for (int i = 1; i < N; i++)
          if (a[i-1] < a[i])
          {
             int t = a[i];
             a[i] = a[i-1];
             a[i-1] = t;
          }

      for (int i = 0; i < N; i++)
          StdOut.print(a[i] + " ");
      StdOut.println();
    }
}
```

Each line of text on the left-hand side below represents the standard input for a run of `ArrayQ`. In this problem, you must determine the output for each of these runs. On the right-hand side, labeled from **A** to **F**, the correct outputs are given, but the order of the lines on the left is not the same as the order of the lines on the right. Indicate which output matches each input by writing the corresponding letter in each box. Each letter should appear in exactly one box.

6 5 4 3 2 1	C	**A.**	5 4 6 3 2 1
1 3 5 7 2 4	F	**B.**	2 3 4 5 6 1
3 6 4 5 1 2	D	**C.**	6 5 4 3 2 1
6 3 1 5 2 4	E	**D.**	6 4 5 3 2 1
5 3 4 6 1 2	A	**E.**	6 3 5 2 4 1
1 2 3 4 5 6	B	**F.**	3 5 7 2 4 1

4. Redirection and piping (6 points). Consider the following Java program:

```
public class EZ
{
   public static void main(String[] args)
   {
      int x = StdIn.readInt();
      int y = Integer.parseInt(args[0]);
      StdOut.print(x + y + " ");
      StdOut.print(x + " ");
      StdOut.println(y);
   }
}
```

The file `input.txt` contains the following text:

```
6 5 4 3
```

A. Write the result of executing the following command. Circle your answer.

```
java EZ 3 < input.txt
```

x becomes 6; y becomes 3; the first print does integer addition before autopromotion from int to String and String concatenation.

```
9 6 3
```

B. Write the result of executing the following command. Circle your answer.

```
java EZ 3 < input.txt | java EZ 2
```

The first program's output becomes the second program's input. So in the second program: x becomes 9; y becomes 2; and again integer addition before autopromotion from int to String and String concatenation.

```
11 9 2
```

5. Debugging (8 points). Consider the following program, which is intended to compute the mean of *N* integers from standard input, **rounded to the nearest integer,** where *N* is a command-line argument.

```
 1      public class Mean
 2      {
 3          public static void main(String[] args)
 4          {
 5             int N = args[0];

 6             int[] a = new int[N];
 7             for(int i = 0; i < N; i++)
 8                a[i] = StdIn.readInt();

 9             int sum = a[0];
10             for(int i = 1; i <= N; i++)
11                sum = sum + a[i];

12             StdOut.println("Mean: " + sum/N);

13          }
14      }
```

In the spaces provided below, identify (*in ten words or less*, each) three bugs and a performance problem in this code. Use the line numbers to refer to the code, and circle your answers. *You do not have to write code to fix the bugs.*

A. A bug that will keep the code from compiling:
5: cannot store String args[0] in an int.
Example fix: int N = Integer.parseInt(args[0]);

B. A bug that will cause the program to crash at runtime:
10-11: a[i] out of bounds when i==N
Example fix: loop i < N rather than i <= N.

C. A bug that will cause the program to print the wrong answer on many inputs:
12: sum/N integer division truncates rather than rounding.
Example fix, casting to double to do division and then rounding to nearest integer:
Math.round((double) sum / N)

D. What can be done to reduce the program's memory usage?
6-11: keep running sum from input -- without array.
Example fix, replacing lines 6-11 with:

```
int sum = 0;
for(int i = 0; i  < N; i++)
   sum += StdIn.readInt();
```

6. Performance (6 points). A Java programmer experiences the following approximate running times for a program that reads a digital photo from an *N*-megabyte file, for various values of *N*.

N	*time*
1	just over 30 seconds
5	just under 3 minutes
50	about half an hour
100	?

A. In the blank at right, give an estimate of the running time for $N = 100$: _about 1 hour_

B. Which of the following best describes the order of growth of the running time of this program as a function of the size of its input? Circle your answer.

a. ~~Logarithmic~~

b. Linear

c. ~~Linearithmic~~

d. ~~Quadratic~~

e. ~~Cubic~~

C. The photo's aspect ratio is 4:3, so the program represents it as a two-dimensional array of size $4R$ by $3R$ (one entry per pixel), where the parameter R depends on the resolution of the photo. The program does some image processing that involves a constant amount of time per pixel. Which of the following best describes the order of growth of the running time of this program as a function of R? Circle your answer.

a. ~~Logarithmic~~

b. ~~Linear~~

c. ~~Linearithmic~~

d. Quadratic

e. ~~Cubic~~

7. Methods (12 points). Consider the following program.

```
public class PassByValueAndScopeQ
{
   public static void f(int[] a, int m)
   {
      double[] b = f(a, 6.0);
      a[0] = a[0] * m;
      a[2] = (int) (b[0] + b[1]);
      m = m + 5;
   }
   public static double[] f(int[] a, double x)
   {
      double[] b = new double[2];
      b[0] = a[0] + x;
      b[1] = a[1] + x;
      return b;
   }
   public static void main(String[] args)
   {
      int[] a = {2, 3, 4};
      int k = 5;
      f(a, k);
      StdOut.println(k);
      StdOut.println(a[0]);
      StdOut.println(a[1]);
      StdOut.println(a[2]);
   }
}
```

Give the output of this program in the blanks provided below.

A. First line: 5

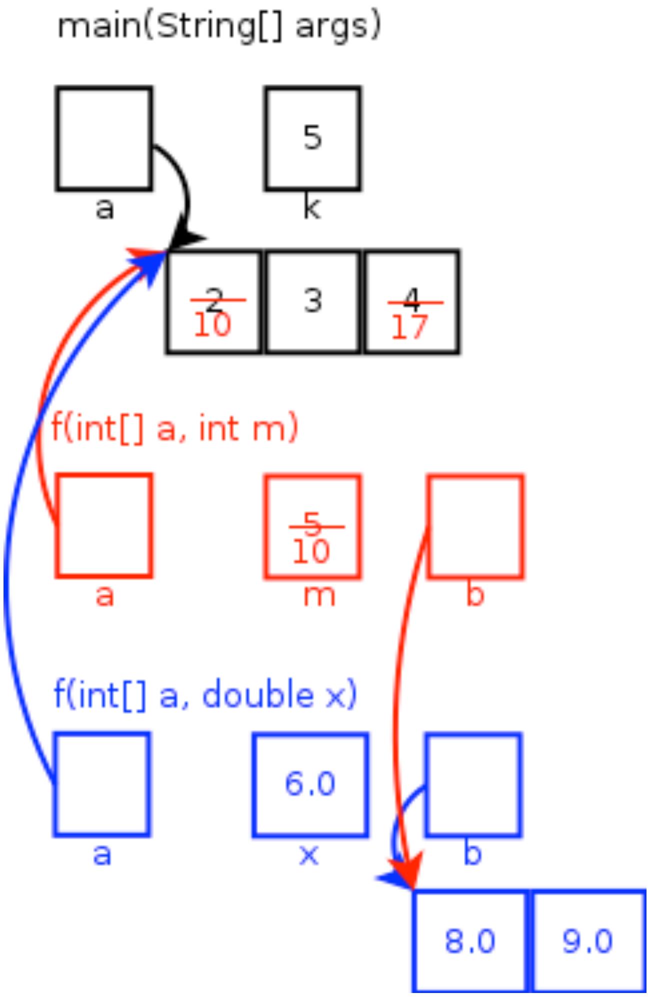
B. Second line: 10

C. Third line: 3

D. Fourth line: 17

See next page (blank page from exam) for memory schematic trace through the program.

This page intentionally left blank. The exam continues on the next page.



8. Recursion (10 points). Consider the following program, which includes a recursive method:

```
public class CrazyR
{
   public static void R(int n, int t)
   {
      if (n == 0)
      {
         StdOut.print(t + " ");
         return;
      }
      R(n-1, 3*t);
      R(n-1, 3*t+2);
      R(n-1, 3*t+1);
   }

   public static void main(String[] args)
   {
      R(2, 0);
      StdOut.println();
   }
}
```

A. Give the output generated when this program is compiled and run. *Hint*: Draw the tree showing the recursive calls in the space provided, then write the output in the underlined spaces provided at the bottom.

```
                                (2,0)
      (1,0)                     (1,2)                      (1,1)
(0,0) (0,2) (0,1)     (0,6) (0,8) (0,7)     (0,3) (0,5) (0,4)
```

Each method at level n==0 outputs its value of t.

The methods at level n==0 are executed in the order shown (left to right), resulting in the following output:

Output: _0_ _2_ _1_ _6_ _8_ _7_ _3_ _5_ _4_

(This problem continues on the next page.)

(Continued from the previous page.)

B. What is the number of numbers the program would print,
if the call to `R` in `main()` were changed to `R(5, 2)`? ____243___

Each call of R(0, ...) prints 1 number.
Each call of R(y, ...) for y > 0 makes 3 calls R(y-1, ...).
So the tree rooted at R(y, ...) has 3^y "leaf" nodes -- calls of R(0, ...).
Thus, the tree rooted at R(5, 2) has 3^5 leaf nodes, and prints 3^5 = 243 numbers.

C. What is the *last* number printed for the call `R(5, 2)`? ___607____

The last number printed is printed in the leaf that is reached by traversing from the root down to the leaf level, taking the 3rd -- R(n-1, 3*t+1) -- call each time:

R(5,2)
R(4, 7)
R(3, 22)
R(2, 67)
R(1, 202)
R(0, 607)