CSE 131 Introduction to Computer Science **Fall 2013**

Exam I

Given: 30 September 2013 Due: End of session

This exam is closed-book, closed-notes, no electronic devices allowed. The exception is the "cheat sheet" on which you may have notes to consult during the exam. Answer questions on the pages of the exam. Do not unstaple the pages of this exam, nor should you attach any other pages to the exam. You are welcome to use the blank space of the exam for any scratch work.

Your work must be legible. Work that is difficult to read will receive no credit. Do not dwell over punctuation or exact syntax in code; however, be sure to indent your code to show its structure.

You must sign the pledge below for your exam to count. Any cheating will cause the students involved to receive an F for this course. Other action may be taken. If you need to leave the room for any reason prior to turning in your exam, you must give your exam and any electronic devices with a proctor.

You must fill in your identifying information correctly. Failure to do so is grounds for a zero on this exam. When you reach this point in the instructions, please give the instructor or one of the proctors a meaningful glance.

Print clearly the following information:		
Name (print clearly):		
Student 6-digit ID (print *really* clearly):		
What time do you attend studio/lab?		
What room (218, 222, or 214)? your best guess		
Problem Number	Possible Points	Received Points
1	20	
2	30	
3	25	
4	25	
Total	100	

Pledge: On my honor, I have neither given nor received any unauthorized aid on this exam.

Signed: _____

(Be sure you filled in your information in the box above!)

1. (20 points)

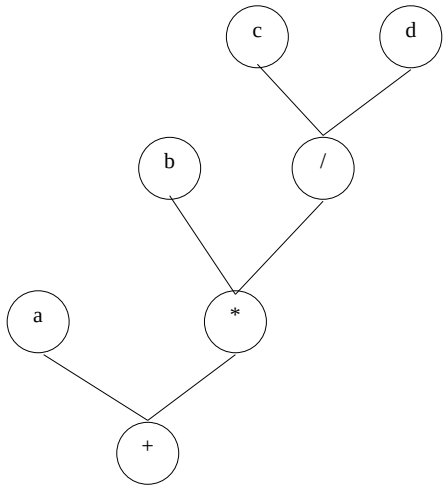
   (a) (10 points) Circle the correct type for each expression in the table below, and state the result of evaluating the expression:

Expression	Type	Result
`3.0 / 2`	double int boolean String	
`3 / 2`	double int boolean String	
`131 > 132`	double int boolean String	
`1.0 + "0"`	double int boolean String	
`true && false`	double int boolean String	
`"100" + (30+1)`	double int boolean String	
`"p" + (1/2) + "wned"`	double int boolean String	
`true || false`	double int boolean String	
`3/2 <= 1`	double int boolean String	
`! (false || (1 < 2))`	double int boolean String	

   (b) (5 points) Below draw the expression tree for the expression

$$1 * 2 + 1 + 2 * 2$$

(c) (5 points)



Complete the blanks below regarding the tree shown above, which uses the arithmethic operators +, ∗, and /:

- The _____ operator is the first operation to execute.
- The _____ operator is the last operation to execute.

2. (30 points)

(a) (10 points) Complete the code below so that it prints `true` if all three of the variables have different values (*i.e.*, no two have the same value). Otherwise it should print `false`.

```
int a = ap.nextInt("Value for a?");
int b = ap.nextInt("Value for b?");
int c = ap.nextInt("Value for c?");
```

Continued on next page...

(b) (10 points) Complete the code below so that it prints the average of `N` random numbers, with each random number chosen by a call to `Math.random()`. Do not use any arrays!

```
int N = ap.nextInt("How many random numbers?");
```

(c) (10 points) Complete the code below so that it prints the maximum of `N` random numbers, with each random number chosen by a call to `Math.random()`. Do not use any arrays!

```
int N = ap.nextInt("How many random numbers?");
```

3. (25 points) Consider the following code, which is the basis for all parts of this question:

```
int N = ap.nextInt();  // prompt for N
double[] nums = new double[N];
for (int i=0; i < N; ++i) {
   double r = 2 * Math.random();
   nums[i] = r;
}
```

(a) (8 points) Consider the following values:

$$-2.0 \qquad -1.0 \qquad -0.5 \qquad 0.0 \qquad 0.5 \qquad 1.0 \qquad 1.5 \qquad 2.0$$

Circle all of the values above that could possibly be assigned to `r` as the above code executes.

Continued on next page...

(b) (12 points) Suppose we are interested in the number of values placed in the array `nums` that have value at or below 0.5. This array has already been computed, as shown on the preceding page. Do not recompute it!

Below, you write code that executes after the code shown on the previous page. Declare and initialize a variable named `count` and write code below the declaration so that when your code is over, `count` is the number of elements in `nums` that have value at or below 0.5. Do not print anything out yet. For example, if the array had the values 0.6, 0.4, 0.6, and 0.5, then your code should compute `count` as 2.

(c) (5 points) Now suppose that `count` has the proper count of the elements at or below 0.5 in `nums`. Below write code, using multiple lines if needed, that uses `count` and any other data in our program so far to print out the percentage of elements in `nums` that have value at or below 0.5. For full credit, your code should print at most 1 decimal place after the decimal point. For example, if `nums` has 3 values and we determine `count` to be 1, then $\frac{1}{3}$ of the enties are at or below 0.5, and your program should therefore print 33.3.

4. (25 points) Recall that the Sieve of Eratosthenes is a method of computing prime numbers that methodically eliminates numbers known not to be prime. It turns out that Eratosthenes had a lesser-known brother, Bubbatosthenes, who had his own, lesser-known sieve. While Eratosthenes was pondering primes using an algorithm that would regale computer science students long after his death (∼2200 years ago), his brother was obsessed with the number 7, which he felt was unlucky.

   In my sieve, said BubbaT, I shall eliminate all numbers that are multiples of 7. Not only that, I shall eliminate any integer before or after a number that is a multiple of 7. Although 0 is a multiple of 7, BubbaT did not notice that, so he only eliminated positive multiples of 7 (and their neighbors).

   On paper, BubbaT's sieve would commence as follows:

   0 1 2 3 4 5 ~~6~~ ~~7~~ ~~8~~ 9 10 11 12 ~~13~~ ~~14~~ ~~15~~ 16 17 18 19 ~~20~~ ~~21~~ ~~22~~ 23...

   (a) (4 points) Describe the data type you would use to represent BubbaT's sieve.

   (b) (4 points) Based on the data type you have chosen, fill in the blanks below to declare and allocate BubbaT's sieve.

```
int N = ap.nextInt("How many numbers in the sieve?");

____________[]  sieve = new _____________[N];
```

   (c) (5 points) Fill in the blanks below so that your data type initially assumes all the integers in your sieve are OK (none have been eliminated yet):[1]

```
for (_____________________________________________) {

    sieve[_______]  =  _______________;
}
```

Continued on next page...

[1] If you want to use a `while` loop instead of the provided `for` loop, that's fine: just write your loop below what I provided.

(d) (7 points) Now write code that modifies your `sieve` to reflect those numbers BubbaT would eliminate (no scaffolding code is given, so write what you need below):

(e) (5 points) Now write code that uses your computed `sieve` to print the numbers that are not eliminated by BubbaT. For example, your output would begin with

$$0, 1, 2, 3, 4, 5, 9, \ldots$$

Formatting of your output is not important.