COS 126 Written Exam 1 Fall 2018

There are seven questions on this exam, each worth ten points. There is one question per lecture, numbered corresponding to the lectures, not in order of difficulty. If a question seems difficult to you, skip it and come back to it. You will have 50 minutes to complete the exam. **This exam is preprocessed by computer.** If you use pencil (and eraser), write darkly.

Resources. You may reference your optional two-sided 8.5-by-11 handwritten "cheat sheet" during this exam. You may not use the textbook, your notes, or any electronic devices. You may not communicate with anyone except the course staff during this exam.

Discussing this exam. Due to travel for extracurriculars and sports, some of your peers will take this exam later. Do not discuss its contents with anyone who has not taken it.

This page. Do not remove this exam from the exam room. Fill in this page now, but do not start the exam until you are told.

Name

NetID

Precept

Exam Room

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

_____

_____

[copy the pledge here]

_____

[signature]

Q1a. Types and Casts. Put the letter to left of each line that gives the value printed (or represents the indicated error). A letter may be used more than once (and some not at all).

☐	`StdOut.println("4" * 4);`	**A**	`1234`
☐	`StdOut.println(1 + 63.0);`	**B**	`64`
☐	`StdOut.println((int) 32.0 + 32);`	**C**	`4444`
☐	`StdOut.println(1 + 2 + 3 + "4");`	**D**	`64.0`
☐	`StdOut.println(64 + "." + 0);`	**E**	`334`
		F	`163.0`
		G	compile-time error
		H	run-time error

Q1b. Terminology. Put the letter to left of each definition that identifies the term defined. No letter may be used more than once.

☐	A set of values and a set of operations on those values.	**A**	literal
☐	A programming-language representation of a value.	**B**	declaration
☐	A programming-language representation of a type.	**C**	variable
☐	A statement that associates a variable with a type	**D**	data type
☐	A statement that associates a value with a variable	**E**	assignment
		F	algorithm
		G	identifier
		H	none of the above

Q2. Loops and conditionals. Consider the following Java code fragment:

```
int count = 0;
int N = ______; // SEE BELOW
for (int i = 0; i < N; i++)
   if (((i/10) % 2) == 1) count++;
StdOut.println(count);
```

A. (1 point). What value is printed when `N` is `5`?

B. (1 point). What value is printed when `N` is `10`?

C. (2 points). What value is printed when `N` is `20`?

D. (2 points). What value is printed when `N` is `1000`?

E. (2 points). What value is printed when `N` is `1024`?

F. (2 points). What value is printed when `N` is `975`?

Q3. Arrays.

A. (3 points) In the box below, write one line of Java code that creates a 3-by-3-by-3 (three-dimensional) array named `a` of `int` values and initializes them each to the value `0`.

B. (7 points) Indicate whether each of the following statements about Java arrays are true or false by writing **T** or **F** respectively in the box to its left.

☐ Creating a Java array takes time proportional to its length.

☐ If `a`, `b`, and `c` are arrays of length `N`, the statement `c = a + b` is equivalent to the statement `for (i = 0; i < N; i++) c[i] = a[i] + b[i]`.

☐ Once a one-dimensional array has been created and initialized, you can refer to the first element in the array with the code `a[0]`.

☐ Java arrays enable programmers to mix different types of data in the same data structure.

☐ It is reasonable for a programmer to expect that the time taken to access an element using its index is independent of the array length.

☐ If `a` and `b` are arrays of the same length, then the code `a = b` copies each element of `b` to the corresponding position in `a`.

☐ The number of items in an array `a` is `a.length -1`.

Q4. I/O. Consider the following Java program, which has no purpose other than to test your understanding of standard input and standard output:

```
public class Q4
{
   public static void main(String[] args)
   {
       while (!StdIn.isEmpty())
       {
          int N = StdIn.readInt();
          int sum = 0;
          for (int i = 0; i < N; i++)
              if (!StdIn.isEmpty())
                  sum += StdIn.readInt();
          StdOut.print(sum + " ");
       }
       StdOut.println();
    }
}
```

Suppose that the file `dataQ4` contains "`1 2 3 4 5 6 7 8 9`". In the box to the left of each command, write the output that it produces. For brevity, we say "`java`" which you may read as "`java-introcs`". If the command results in an error or produces no output, write **NONE**.

```
% java Q4 dataQ4
```

```
% java Q4 | dataQ4
```

```
% java Q4 < dataQ4
```

```
% java Q4 < dataQ4 | java Q4
```

```
% java Q4 < dataQ4 | java Q4 | java Q4
```

Q5. Functions. Our task is to develop a function for computing the absolute value of each of the elements in a 1-dimensional array of `int` values. For example, for the array `{1, -6, 2, -1}` the desired result is `{1, 6, 2, 1}`. The code below is a solution to this problem, with five snippets of code missing. Your task is to identify the code needed to complete this program. We explore two different design patterns.

```
public static ONE abs(int[] x)
{
     int N = x.length;
     TWO
     for (int i = 0; i < N; i++)
        if (x[i] < 0)
              THREE
        FOUR
    FIVE
}
```

A. (5 points) Suppose that the design is to modify the argument array to reflect the result. Using the table at right, fill in each box with the letter corresponding to the code that is needed to accomplish this.

☐ ONE ☐ FOUR

☐ TWO ☐ FIVE

☐ THREE

B. (5 points) Suppose that the design is to return a new array that reflects the result. Again using the table at right, fill in each box with the letter corresponding to the code that is needed to accomplish this.

☐ ONE ☐ FOUR

☐ TWO ☐ FIVE

☐ THREE

A	`return x;`
B	`return z;`
C	`x[i] = -x[i];`
D	`int[] z;`
E	`z[i] = -x[i];`
F	`int[] z = new int[];`
G	`new int[N]`
H	`int[] z = new int[N];`
I	`else z[i] = x[i];`
J	`void`
K	`int`
L	`int[]`
M	`x[i] = x[i];`
N	`new int[]`
O	`else z[i] = -x[i];`
P	no code for this box

Q6. Recursion. Consider the following recursive static method:

```
static int f(int n)
{
   if (n == 0) return 0;
   if (n == 1) return 0;
   if (n == 2) return 1;
   return f(n-1) + f(n-2) - f(n-3);
}
```

A. (1 point). What is the value of `f(3)` ?

B. (1 point). What is the value of `f(4)` ?

C. (1 point). What is the value of `f(7)` ?

D. (3 points). Including the first one, how many calls on `f()` are made to compute `f(7)`?

E. (2 points). What is the value of `f(101)` ?

F. (2 points). Write T in the box at right if you could compute the value of `f(101)` using this code in less than 10 seconds on your computer, otherwise write F.

Q7. TOY. Consider the following TOY program:

```
10:  7100  R[1] <- 0000
11:  8FFF  read R[F]
12:  9F15  M[15] <- R[F]
13:  82FF  read R[2]
14:  1112  R[1] = R[1] + R[2]
15:  C016  goto 16
16:  91FF  write R[1]
17:  0000  halt
```

Fill in the following table to give the value(s) printed on standard output for each given sequence of values on standard input. If no output, write **NONE**.

`0000 1112`	
`1112 0000`	
`1112 1112`	
`C011 1112 1112 1112`	
`C011 C011 1112 1112`	

TOY REFERENCE CARD

INSTRUCTION FORMATS

```
             | . . . . | . . . . | . . . . | . . . .|
  Format RR: | opcode  |    d    |    s    |    t   |  (0-6, A-B)
  Format A:  | opcode  |    d    |       addr       |  (7-9, C-F)
```

ARITHMETIC and LOGICAL operations

1:	add	R[d] <- R[s] + R[t]
2:	subtract	R[d] <- R[s] - R[t]
3:	and	R[d] <- R[s] & R[t]
4:	xor	R[d] <- R[s] ^ R[t]
5:	shift left	R[d] <- R[s] << R[t]
6:	shift right	R[d] <- R[s] >> R[t]

TRANSFER between registers and memory

7:	load address	R[d] <- addr
8:	load	R[d] <- M[addr]
9:	store	M[addr] <- R[d]
A:	load indirect	R[d] <- M[R[t]]
B:	store indirect	M[R[t]] <- R[d]

CONTROL

0:	halt	halt
C:	branch zero	if (R[d] == 0) PC <- addr
D:	branch positive	if (R[d] > 0) PC <- addr
E:	jump register	PC <- R[d]
F:	jump and link	R[d] <- PC; PC <- addr

Register 0 always reads 0.
Loads from M[FF] come from stdin.
Stores to M[FF] go to stdout.

16-bit registers (using two's complement arithmetic)
16-bit memory locations
8-bit program counter

This page is provided as scratch paper. If you tear it out, write your name, NetID, and precept below and return it inside your exam.

Name _____ NetID _____ Precept_____