COS 126	Intro to CS	Fall 2014

Midterm 2 Written Exam

This test has 10 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a one page cheatsheet (front and back). 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 has taken the exam. You have 50 minutes to complete the test.

This exam is preprocessed by computer: use a pen (or make sure your answers with a pencil are not faint) and do not write any answers outside of the designated frames.

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

Name:

netID:

Room:

	P01	P01A	P01B	P02	P02A	P02B	P02C	P03	P03A	P04	P05	P05A	P05B	P06	P06A
Pr:	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

P01	T/Th 12:30pm	Donna Gabai
P01A	T/Th 12:30pm	Maia Ginsburg
P01B	T/Th 12:30pm	Andrea LaPaugh
P02	T/Th 1:30pm	Dan Leyzberg
P02A	T/Th 1:30pm	Stephen Cook
P02B	T/Th 1:30pm	Andrea LaPaugh
P02C	T/Th 1:30pm	Jordan Ash
P03	T/Th 2:30pm	Dan Leyzberg
P03A	T/Th 2:30pm	Shaoqing (Victor) Yang
P04	T/Th 7:30pm	Katie Edwards
P05	W/F 1:30pm	Doug Clark
P05A	W/F 1:30pm	Maia Ginsburg
P05B	W/F 1:30pm	Ted Brundage
P06	W/F 2:30pm	Doug Clark
P06A	W/F 2:30pm	Donna Gabai

1. Number systems (7 points).

A. (3 points) Fill in the missing entries in the following table by converting the numbers between bases.

hexadecimal	*binary*	*decimal*
8A	10001010	138
2D	00101101	45
3A	0011 1010	58
12	0001 0010	18

B. (4 points) Assume that x is a two's complement binary integer whose absolute value is not large (so no overflow problems). At right are four Java expressions that use the << (shift left) and ∼ (bitwise complement: flip each 0 to 1 and each 1 to 0 in the binary representation) operators. For example, `7 << 1` is `14` and `~0` is `-1`. Match each expression to one of the mathematical functions at left (the one that it computes), by writing a, b, c, or d in each blank. You must use each letter exactly once and you must fill in all the blanks.

a. $4x$

b. $-x$

c. $3x$

d. $-x - 1$

b	`(~x) + 1`
c	`(x << 1) + x`
d	`~x`
a	`x << 2`

2. Programming Languages (5 points).

For each of the following statements, mark True if it applies, and False if it does not. **For each line, only fill in *one* answer.**

A. A memory leak is when...

	True	False
... a function is passed to another function.	○	●
... a Java program reuses an array variable.	○	●
... a Python program allocates a lot of memory for its own objects.	○	●
... a C/C++ program does not "free" memory that it allocated.	●	○
... a recursive function has no base case.	○	●

B. Compile-time type checking...

	True	False
... indicates the lack of a garbage collector.	○	●
... makes Java programming more difficult.	○	●
... cannot happen if you use generics.	○	●
... is not possible in Python.	●	○
... is a central feature of Matlab.	○	●

C. Functional programming languages...

	True	False
... typically allow compact code for powerful operations like map and reduce.	●	○
... generally do not support compile-time type checking.	○	●
... do not include Python.	○	●
... allow functions to take functions as arguments.	●	○
... do not support recursion.	○	●

3. REs (7 points). Let $L = \{ab, aaab, aaaab, aabaab, aabaaab\}$. For each of the regular expressions below *check the only answer that applies.*

The possible options (and their shortnames) are:

- [NONE] Matches no strings in L.
- [SOME] Matches only some strings in L and some other strings.
- [MORE] Matches all strings in L and some other strings.
- [EXACT] Matches all strings in L and no other strings.

	NONE	SOME	MORE	EXACT
A. `(aa*b)*`	○	○	●	○
B. `a*b*`	○	●	○	○
C. `(a|b)*ab`	○	○	●	○
D. `a*baba*b*`	●	○	○	○
E. `(ab)|(a(a|aba)(a|aa)b)`	○	○	○	●
F. `a*baa*b*`	○	●	○	○
G. `(a|(aaa)|(aaaa))b|(aabaa(b|ab))`	○	○	○	●

4. Linked structures (7 points). Examine the following code and answer the questions below:

```
public class Node
{
   private String name;
   private Node next;
   public Node (String s, Node n)
   {  name = s; next = n;  }

   public static void mystery(Node first)
   {
      if (first == null) return; // base case
      mystery(first.next);
      System.out.println(first.name);
   }

   public static void main(String[] args)
   {
      Node a = new Node("Alice", null);
      Node b = new Node("Bob", a);
      Node e = new Node("Eve", a);
      Node t = a;
      t.next = e;
      t.next.next = b;
      b.next = null;
      mystery(t);
   }
}
```

A. (1 point) Check the value of the indicated node reference after completion of the *first three statements* (which create the nodes) in `main()`.

	a	b	e	null
a.next	○	○	○	●
b.next	●	○	○	○
e.next	●	○	○	○

B. (3 points) Check the value of the indicated node reference after `main()` completes execution.

	a	b	e	null
a.next	○	○	●	○
b.next	○	○	○	●
e.next	○	●	○	○

C. (3 points) Give the lines of output printed by the program.

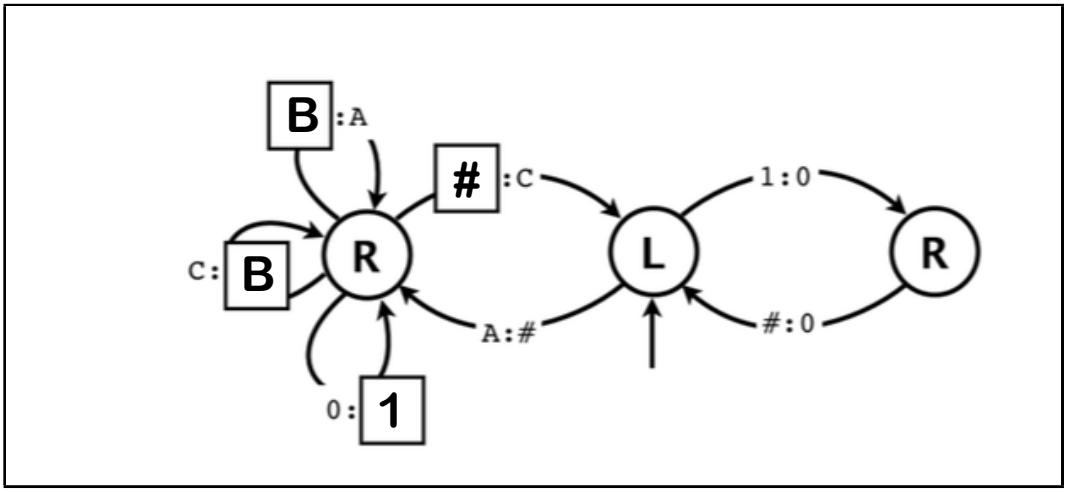
Bob
Eve
Alice

5. Turing Machines (8 points).

The incomplete Turing machine below is supposed to write onto its tape the Fibonacci sequence 1, 1, 2, 3, 5, 8, 13, ..., but in unary notation: `1`, `1`, `11`, `111`, `11111`, `11111111`, `1111111111111`. You will recall that each Fibonacci number is the sum of the previous two. This Turing machine uses the symbols `A`, `B`, and `C` to delineate the two most recently computed Fibonacci numbers. As usual, we omit transitions to the same state that do not change the symbol. Here is the tape before computing Fibonacci number 5, with the tape head positioned at the `C`:

		.	.	.	#	#	#	1	#	1	A	1	1	B	1	1	1	C	#	#	#	.	.	.		

Fill in exactly one symbol in each of the 4 empty squares below to complete the design of this Turing machine. To avoid worrying about initial conditions, you may assume that the initial contents of the tape and the position of the tape head are as given above and that the machine starts in the middle state, as indicated. Do not add new states or new transitions, and do not use tape symbols other than `#`, `0`, `1`, `A`, `B`, or `C`.



6. Sorting (7 points). Describe the order of growth of the running time of each specified algorithm below on a file of size n.

	n	n^2	$n \log n$
A. Insertion sort for a randomly ordered file	○	●	○
B. Mergesort for a randomly ordered file	○	○	●
C. Building a BST for a randomly ordered file	○	○	●
D. Insertion sort for a file that is in reverse order	○	●	○
E. Insertion sort for a file that is already in order	●	○	○
F. Mergesort for a file that is already in order	○	○	●
G. Building a BST for a file that is already in order	○	●	○

TOY REFERENCE CARD

INSTRUCTION FORMATS

```
             | . . . . | . . . . | . . . . | . . . . |
  Format 1:  | opcode  |    d    |    s    |    t    |  (0-6, A-B)
  Format 2:  | 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] <- mem[addr]
    9: store            mem[addr] <- R[d]
    A: load indirect    R[d] <- mem[R[t]]
    B: store indirect   mem[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 mem[FF] come from stdin.
Stores to mem[FF] go to stdout.

7. TOY (8 points). Consider what happens when the following TOY program is executed by pressing RUN with the program counter set to `10`:

```
10: 7201    R[2] <- 01
11: 7301    R[3] <- 01
12: 8115    R[1] <- Mem[15]
13: C117    if (R[1] == 0)  PC <-  17
14: 2112    R[1] <- R[1] - R[2]
15: 1332    R[3] <- R[3] + R[2]
16: C013    PC <- 13
17: 0000    Halt
```

A. (2 points) What is the value of `R[1]` after the instruction at location `12` completes? Write your 4-digit hexadecimal answer in the blank below.

Answer: **1332**

B. (3 points) What is the value of `R[3]` after the first time the instruction at location `13` completes? Write your 4-digit hexadecimal answer in the blank below.

Answer: **0001**

C. (3 points) What is the value of `R[3]` when the program halts? Write your 4-digit hexadecimal answer in the blank below.

Answer: **1333**

8. Computability/Intractability (8 points). For each of the computational problems below, indicate its difficulty by marking the most appropriate choice among *True*, *False* or *Nobody Knows*.

	True	False	Nobody Knows
A. Every problem in NP is also in P.	○	○	●
B. There is a DFA that can recognize all binary palindromes (binary strings that read the same forwards and backwards).	○	●	○
C. There is a Turing machine that can decide whether the number of 1s on its input tape is prime.	●	○	○
D. The Halting Problem is NP-complete.	○	●	○
E. The Traveling Salesperson Problem is NP-complete.	●	○	○
F. There exists a deterministic Turing machine that can solve every problem in NP.	●	○	○
G. There is a DFA that can recognize the set of all binary strings that contain at least one million 0s and at least one million 1s.	●	○	○
H. If P = NP there is a polynomial-time algorithm for factoring.	●	○	○

9. Boolean algebra and combinational circuits (8 points). The *even parity* function of N Boolean variables is 1 if and only if the number of variables with value 1 is even (including 0).

A. (3 points) Fill in the missing entries in this truth table for the 3-variable *even parity* function.

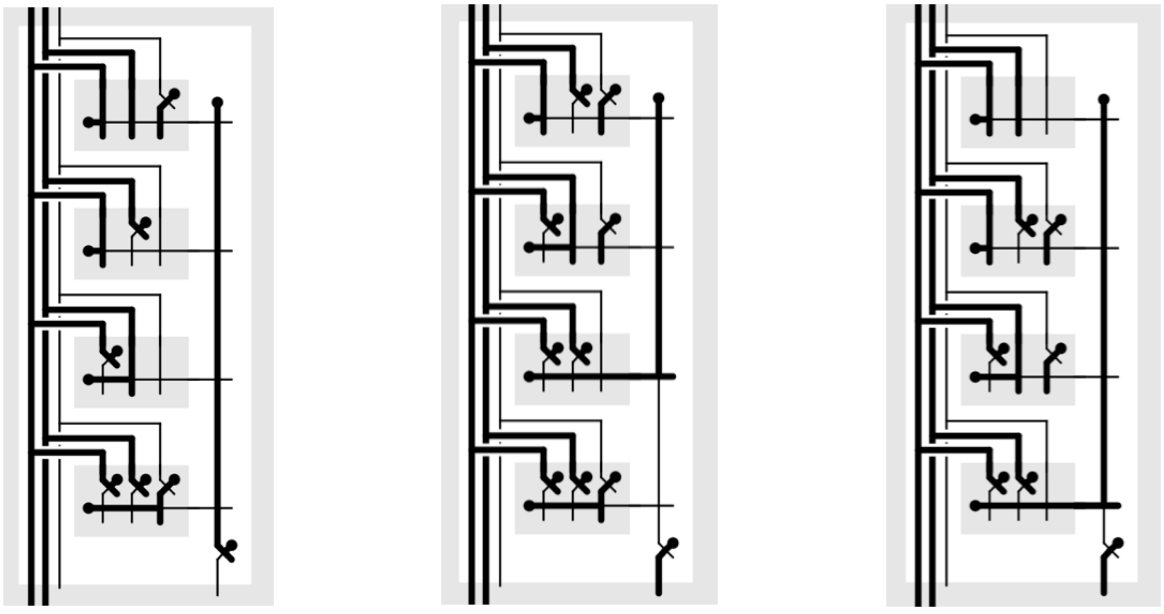
x	y	z	*even parity*
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

B. (3 points) In the box, write out the sum-of products form of *even parity* for 3 variables.

x'y'z' + x'yz + xy'z + xyz'

C. (2 points) Which of the circuits below is computing *even parity* for 3 variables with the inputs 1 1 0? In each circuit, assume that the inputs `xyz` are provided in that order to the three lines at the upper left and the output is the line at the bottom right. Check your answer in the circle above the correct circuit.

○ ○ ●



10. CPU (5 points). Identify each of the CPU components below as either a *combinational circuit* or a *sequential circuit*.

	Combinational Circuit	Sequential Circuit
A. ALU	●	○
B. MUX	●	○
C. Register	○	●
D. IR	○	●
E. Control	●	○
F. PC	○	●
G. Incrementer	●	○
H. Memory	○	●