COS 126	General Computer Science	Spring 2014

Written Exam 2

This exam is closed book, except that you are allowed to use a one-page double-sided cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided.

Print your name, netID, lecture number and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. It is a violation of the Honor Code to discuss this exam until everyone in the class has taken the exam. You have 50 minutes to complete the test.

Write out and sign the Honor Code pledge before turning in the test:
"I pledge my honor that I have not violated the Honor Code during this examination."

Pledge: ____

____

Signature: ____________________

Name: ____________________

NetID: ____________________

Registered Lecture: ____________________

Precept: ____________________

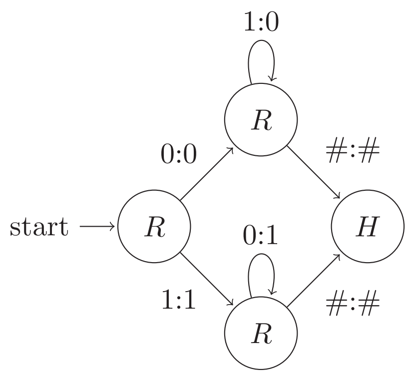
L01	10:00 TTh	
L02	11:00 TTh	
P01	12:30 TTh	David Pritchard
P01A	12:30 TTh	Donna Gabai
P01B	12:30 TTh	Aleksey Boyko
P02	1:30 TTh	Borislav Hristov
P02A	1:30 TTh	Terry Yannan Wang
P02B	1:30 TTh	Aleksey Boyko
P02C	1:30 TTh	Nanxi Kang
P02D	1:30 TTh	Xinyi Fan
P03	2:30 TTh	Borislav Hristov
P03A	2:30 TTh	Bebe Shi
P04	3:30 TTh	Kevin Lee
P04A	3:30 TTh	Victor Shaoqing Yang
P05	7:30 TTh	Kevin Lee
P06	10:00 WF	Mojgan Ghasemi
P07	11:00 WF	Mojgan Ghasemi
P08	12:30 WF	Donna Gabai
P08A	12:30 WF	Maia Ginsburg
P09	1:30 WF	David Pritchard
P09A	1:30 WF	Maia Ginsburg
P09B	1:30 WF	Judi Israel
P10	2:30 WF	Judi Israel

Problem	**Value**	**Score**
*0	1	
1	6	
2	12	
3	12	
4	6	
5	6	
6	10	
7	8	
8	9	
Total	**70**	

* Question 0: Did you show up to the right room at the right time?

1 Turing Machines (6 points)

Consider the following Turing Machine. *Remember that for any transition not otherwise indicated, you should read and write the same symbol while returning to the same state.*



(a) Suppose we execute this Turing Machine on the tape given below. What are the final tape contents? Drawing some intermediate steps is optional.

Initial tape contents:	⋯	#	0	1	1	0	0	1	0	1	0	1	#	⋯
Initial head location:		↑												

Final tape contents:	⋯	#	0	0	0	0	0	0	0	0	0	0	#	⋯

(b) Repeat part (a) with this initial tape.

Initial tape contents:	⋯	#	1	0	1	0	1	1	0	0	1	1	#	⋯
Initial head location:		↑												

Final tape contents:	⋯	#	1	1	1	1	1	1	1	1	1	1	#	⋯

(c) Describe in English, using 15 words or less, what this Turing Machine does in general:

Description: Copies the leftmost digit over all other digits.

2 Universality, Computability, and Intractability (12 points)

For parts (a)–(e), determine whether the given statement is true or false, and circle the appropriate answer.

(a) Any set of strings that can be described by a regular expression can be recognized by a Turing Machine.

true **false**

(b) Any set of strings that can be recognized by a Turing Machine can be described by a regular expression.

true **false**

(c) The undecidability of the Halting Problem means that P does not equal NP.

true **false**

(d) If someone proves that factoring has a polynomial-time algorithm, then P=NP.

true **false**

(e) If someone proves that P=NP, then factoring has a polynomial-time algorithm.

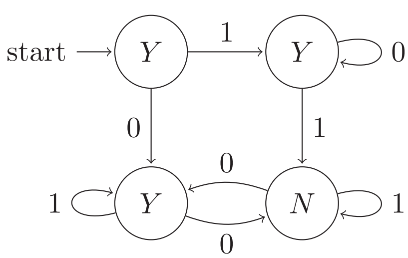
true **false**

(f) Which two of the following orders of growth are polynomial? *Circle exactly two.*

$N^{\log N}$ $10^{\log N}$ 126^N N^{126}

3 RE/DFA (12 points)

This table has a DFA or an RE in each row, and a string at the top of each column. Determine whether each RE matches each string, and whether the DFA accepts each string. *Circle the correct choice in each box.*

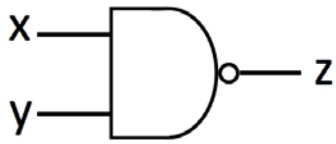
	`00000`	`01111`	`11111`
$\left(0(0\|1)^*0\right) \mid \left(1(0\|1)^*1\right)$	Matches Doesn't Match	Matches Doesn't Match	Matches Doesn't Match
	Accepts Rejects	Accepts Rejects	Accepts Rejects
$1^*\|1^*01^*$	Matches Doesn't Match	Matches Doesn't Match	Matches Doesn't Match

Willy Woodrow claims that every binary string with an odd number of zeroes is accepted by the pictured DFA. Prove him wrong: write a binary string with an odd number of zeroes that the pictured DFA does *not* accept.

Binary String: **101 (other correct solutions also exist)**

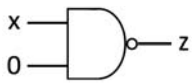
4 Circuits and Boolean Algebra (6 points)

Here is a picture of a "NAND gate", and its truth table:

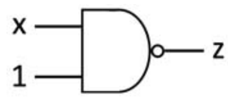


x	y	$z = \text{NAND}(x, y)$
0	0	1
0	1	1
1	0	1
1	1	0

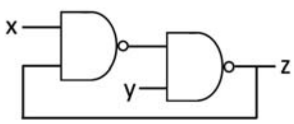
It is the same as $(xy)'$. *For each circuit, enter the letter of the function it performs, taken from the list below. Some letters may be used twice or not at all.*



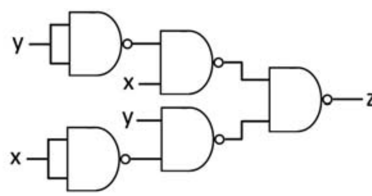
C



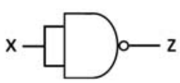
I



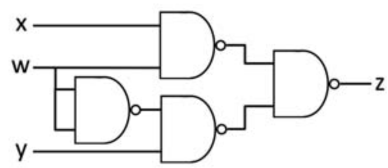
G



E



I



H

A. $z = w + x + y$

B. $z = 0$

C. $z = 1$

D. $z = xy$

E. $z = x \wedge y$ (xor)

F. $z = \text{MAJORITY}(w, x, y)$

G. stores one bit of memory

H. $z = wx + w'y$ (multiplexer)

I. $z = x'$

J. $z = (x \wedge y)'$

5 Data Structures (6 points)

You read the following integers from standard input. As each one is read, you insert it into a data structure named `container`.

`4 2 5 3 1`

Here are three different cases.

Case 1: `container` is a `Queue<Integer>`. After the insertions we run

```
while (!container.isEmpty()) StdOut.print(container.dequeue()+" ");
```

What is the output?

Output: **4 2 5 3 1**

Case 2: `container` is a `Stack<Integer>`. After the insertions we run

```
while (!container.isEmpty()) StdOut.print(container.pop()+" ");
```

What is the output?

Output: **1 3 5 2 4**

Case 3: `container` is a binary search tree. Draw the final tree after all the values are inserted. *Please circle the final tree.*

```
      4
     / \
    2   5
   / \
  1   3
```

6 Abstract Data Types (10 points)

Below we describe several data structures that we want to use in a program. How can we implement these structures in Java? For each, write down the most appropriate data type of the form

- `Stack<T>` or
- `Queue<T>` or
- `ST<K,V>` (symbol table)

where each type parameter `T`, `K`, `V` is one of

- `String` or
- `Double` or
- `Integer` or
- `String[]` or
- `int[]`

For example, the correct answer for "A data type to convert state abbreviations into full state names, like `NJ` to `New Jersey`" would be `ST<String, String>`. Some types may be used multiple times or not at all. Note: your "netid" is like your email address without `@princeton.edu`.

(a) The number of times that each student, listed by netid, has posted on Piazza.

ST<String, Integer>

(b) A data structure to implement the Karplus-Strong algorithm for synthesizing guitar string sounds (it is okay if it is not as efficient as a `RingBuffer`).

Queue<Double>

(c) A data type to look up a student's name, given their student number.

ST<Integer, String>

(d) While tracing a Java program that utilizes recursion or other nested method calls, the sequence of method names of the calls that, at this moment, have not yet returned.

Stack<String>

(e) For every score out of 70 on this exam, the student numbers of everyone who got that score.

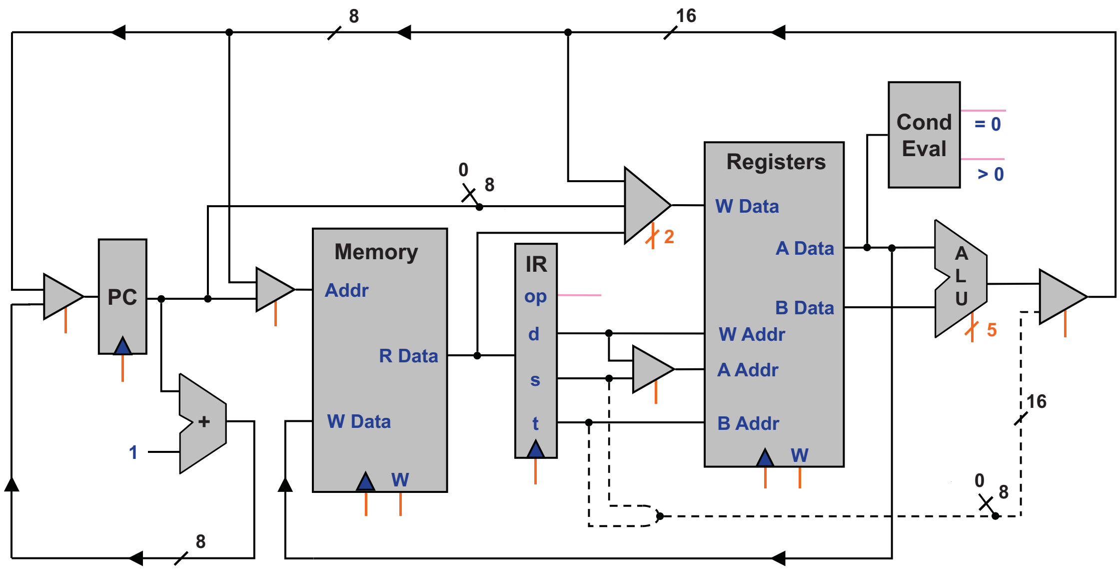
ST<Integer, int[]>

TOY Reference Card *You may use this for the next problem on the facing page.*

```
             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.
pc starts at 10

16-bit registers
16-bit memory locations
 8-bit program counter
```

7 Architecture (8 points)

Below is a datapath diagram of the TOY architecture. You may assume that all control signals work correctly. One data path in the bottom right is shown with dashed lines.



For each opcode in the table below, would it still work correctly if the dashed data path were left out? *Circle the appropriate answer in each row.*

Opcode 1: add	**Would still work**	**Would not work**
Opcode 5: shift left	**Would still work**	**Would not work**
Opcode 7: load address	**Would still work**	**Would not work**
Opcode 8: load	**Would still work**	**Would not work**
Opcode 9: store	**Would still work**	**Would not work**
Opcode A: load indirect	**Would still work**	**Would not work**
Opcode C: branch zero	**Would still work**	**Would not work**
Opcode E: jump register	**Would still work**	**Would not work**

8 Linked Lists (9 points)

A `Trip` data type represents a trip to different cities, starting and ending in the same city, implemented by a circularly linked list. You are using a `Trip` to represent an upcoming vacation of yours. Your friend Joe has planned another `Trip`, starting and ending in one of the cities in your planned `Trip`. Can you find that common city and merge Joe's trip into yours? For example,

```
Your trip : Brooklyn -> Houston -> Toronto -> Princeton -> Brooklyn (==start)
Joe's trip : Toronto -> Charlotte -> Atlanta -> Miami -> Toronto (==start)
The merged trip : Brooklyn -> Houston -> Toronto -> Charlotte -> Atlanta ->
                    Miami -> Toronto -> Princeton -> Brooklyn (==start)
```

The merged trip contains the common city twice (here `Toronto`), and the new distance is the sum of the old distances. Complete a method `travelWith()` to merge Joe's trip into yours. For each of the 4 blanks, pick from one of the lines below, and write the corresponding letter in the blank.

```
public class Trip {
   private class Node {
      private String cityName;
      private Node next;
   }

   private Node start;

   // merge joe's Trip into this trip. assumes both Trips are nonempty.
   public void travelWith(Trip joes) {
        Node node = this.start;
        while (__________________B_______________) { // put a letter in this blank
            node = node.next;
        }
        __________________________J__________;   // put a letter in this blank
        __________________________F__________;   // put a letter in this blank
        __________________________C__________;   // put a letter in this blank
   }
...
}
```

A. `node.cityName.equals(this.start.cityName)`
B. `!node.cityName.equals(joes.start.cityName)`
C. `joes.start.next = tmp`
D. `joes.start.next = node.next`
E. `joes.start = tmp.next`
F. `node.next = joes.start.next`
G. `node.next = joes.start`
H. `tmp = node.next`
I. `tmp = node`
J. `Node tmp = node.next`
K. `Node tmp = node`

PRINT your name here: ______________________________

This page intentionally left blank for scratch paper. Return it with your test.