**NAME:** **login id:**

**Precept:**

# COS 126 Midterm 2 Written Exam Fall 2012

This test has 11 questions, weighted as indicated. The exam is closed book, except that you are allowed to use a two-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 | /5 |
| 2 | /6 |
| 3 | /4 |
| 4 | /6 |
| 5 | /6 |
| 6 | /10 |
| 7 | /9 |
| 8 | /8 |
| 9 | /6 |
| 10 | /5 |
| 11 | /5 |
| | /70 |

1. **Number Systems (5 points).** Java's *bitwise* operators `^`, `&`, and `|` compute the XOR, AND, and OR operations, respectively, on the bits of their arguments: the first bit of `a^b` is the exclusive or of the first bits of `a` and `b`; the second bit is the exclusive or of the second bits of `a` and `b`, and so forth. In the blanks, give the result of performing these operations on the pairs of *hexadecimal* numbers below. Each answer must be a 4-digit hexadecimal number. One of the answers is provided for you.

`FF00^00FF` ____________

`FF00&00FF` ____________

`FF00|00FF` ____________

`1ABC^8654` ____________

`1ABC&8654` ___ **`0214`** ___

`1ABC|8654` ____________

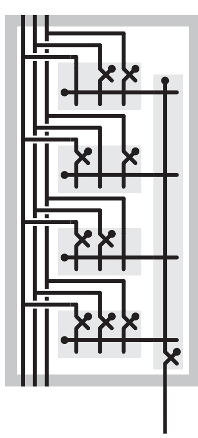
## 2. Boolean Algebra and Combinational Circuits (6 points).

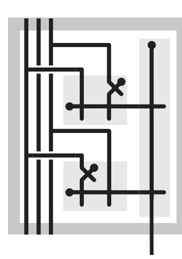
**A.** (2 points) Fill in the truth table below for the Boolean function $p$ of three variables defined as follows: $p$(`x`, `y`, `z`) is true if and only if `xyz` is a *palindrome* (reads the same backwards or forwards). For example, `010` and `000` are palindromes, but `011` is not.

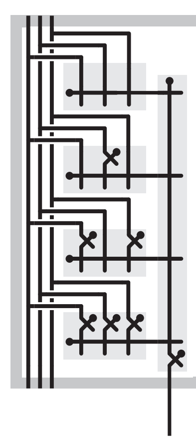
| `x` | `y` | `z` | `p(x, y, z)` |
|---|---|---|---|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

**B.** (2 points) Write out the sum-of products form of $p$.

**C.** (2 points) Which of the circuits below compute $p$? 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 bottom right. Circle your answer(s).





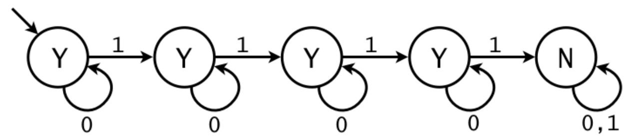


**3. Programming languages (4 points).** Write `YES` or `NO` in each of the eight empty boxes below to indicate whether or not the indicated programming language has the indicated property. One box is filled in for you.

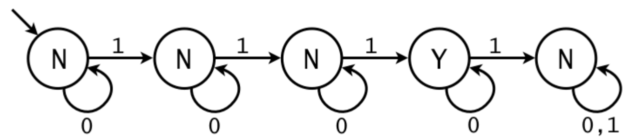
| | compile-time type checking | objects and classes | loops and conditionals |
|---|---|---|---|
| C | | | |
| Java | | | |
| Python | | **`YES`** | |

4. **Regular languages (6 points).** Consider the following four DFAs

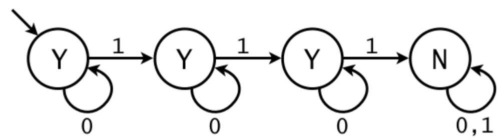
**A.**



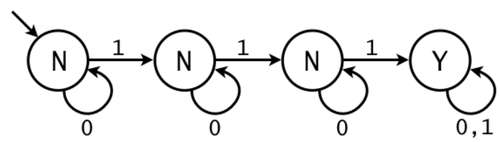

**B.**




**C.**




**D.**




and the following four REs

**1.** `0* | 0*10* | 0*10*10* | 0*10*10*10*`
**2.** `0*(1 | 10*1 | 10*10*1)0*`
**3.** `(0|1)*1(0|1)*1(0|1)*1(0|1)*`
**4.** `0*10*10*10*`

Match these with the languages described in the first column in the following table by writing a letter in each box in the DFA column and a number in each box in the RE column.

| language | DFA | RE |
|---|---|---|
| binary strings with *exactly* three 1s | | |
| binary strings with *at least* three 1s | | |
| binary strings with *at most* three 1s | | |

**5. Stacks and Queues (6 points).** Suppose that we need a `Queue` of `int` values. By mistake, we downloaded `Stack.java` instead of `Queue.java`, so we decide to simulate the queue with a stack and not bother with generics for the queue. Immediately we realize that either `enqueue()` or `dequeue()` must take time proportional to the number of items in the stack, but we decide to live with that because in our application the queue will always be small. Here is a working implementation that is missing 6 lines:

```
public class IntQueue
{
   private Stack<Integer> stack;

   public IntQueue()
   {  stack = new Stack<Integer>();  }

   public void enqueue(int v)
   {
      _____ // missing line 1
      _____ // missing line 2
      _____ // missing line 3
      _____ // missing line 4
      _____ // missing line 5
      _____ // missing line 6
   }

   public int dequeue()
   {  return stack.pop();  }
}
```

Fill in each of the blanks above with one of the six letters below to indicate how to make a working implementation that simulates a `Queue` with a `Stack`. Use each letter exactly once.

**A.** `stack.push(v);`

**B.** `stack.push(tempStack.pop());`

**C.** `Stack<Integer> tempStack = new Stack<Integer>();`

**D.** `tempStack.push(stack.pop());`

**E.** `while (!stack.isEmpty())`

**F.** `while (!tempStack.isEmpty())`

6. **Abstract Data Types (10 points).**

**A.** (6 points) Here are six possible ways to create a symbol table using the generic class `ST<Key, Value>`, where `Key` and `Value` are types.

**A.** `ST<String, ST<String, Integer>>`

**B.** `ST<String, Integer>`

**C.** `ST<Integer, String>`

**D.** `ST<Integer, Queue<Integer>>`

**E.** `ST<String, Queue<Integer>>`

**F.** `ST<Integer, Integer>`

Match each of the six applications to the one of these types of symbol tables by writing the letter of the type on the blank preceding the line describing the application. You should use each letter exactly once.

_________ The sizes, in bytes, of each file in a directory.

_________ The name of the owner of each house number on Nassau Street.

_________ A book's index, listing all page occurrences of each topic.

_________ A table of values of the factorial function.

_________ For each U.S. state, populations of all cities in that state.

_________ All divisors of the first 1000 integers.

**B.** (2 points) Facebook wants to provide users with the facility to look up the friends that we met in each year. For example, this requires a data structure that remembers that in 2011 we met Quinn, in 2009 we met Trey and Mitsy, and in 2007 we met Caitlin. Why is `ST<Integer, String>` problematic for this purpose?

**C.** (2 points) Fill in the code below with types so that the resulting `ST` is suitable for this purpose.

`ST< ____________________ , _________________________ >`

7. **TOY (9 points).** Give a TOY instruction that performs each of the tasks described below. Assume that `R[E]` contains `0001` and that `R[F]` contains `0002` (but make no assumptions about the contents of `R[1]` through `R[D]`). For full credit you must use a different op-code (the first digit) for each answer. It is better to repeat an op-code (for partial credit) than to leave an answer blank. Answers which exactly match a previous answer will be treated as blank. Each answer should be a single 4-digit hexadecimal TOY instruction, written in one of the boxes provided. For your reference, the TOY cheat-sheet is on the next page.

**A, B.** Two ways to double the contents of `R[2]`.

**C, D, E**. Three ways to set `R[2]` to zero.

**F**. Copy `R[2]` to `R[3]`.

**G**. Set `R[3]` to the negative of the value in `R[2]`.

**Reminder:** For full credit, make sure your op-codes above are distinct.

# 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.
```

**8. Name game (8 points).** Match the following names with an associated phrase. Use each letter once and only once.

| | | | |
|---|---|---|---|
| **A.** | Steve Cook | ________ | Stored programs |
| **B.** | Alan Turing | ________ | Reductions |
| **C.** | John von Neumann | ________ | Universality |
| **D.** | James Gosling | ________ | MS Word |
| **E.** | Richard Karp | ________ | SAT is NP-complete |
| **F.** | Alan Kay | ________ | C++ |
| **G.** | Charles Simonyi | ________ | Java |
| **H.** | Bjarne Stroustrup | ________ | Dynabook |

**9. Universality (6 points).** In the blanks provided, mark each of the statements below as true (`T`) or false (`F`).

**A.** ______ A Universal Turing Machine (UTM) can simulate any app on your smartphone.

**B.** ______ If a quantum computer is successfully built, it could provide a counterexample to the Church-Turing thesis.

**C.** ______ A UTM can simulate the operation of any Turing machine, including itself.

**D.** ______ No Turing machine can decide whether a given DFA halts.

**E.** ______ A UTM can decide whether a given string is in the language described by a given regular expression.

**F.** ______ The Church-Turing thesis implies that no computer can solve the halting problem.

10. **Intractability (5 points).** Match each of the statements on the left with the best statement on on the right by writing **1**, **2**, **3**, or **4** in each of the blanks provided.

**A.** ______ Some instances of TSP can be solved in polynomial time on a deterministic Turing machine.

**B.** ______ All instances of 3-SAT can be solved in polynomial time on a deterministic Turing machine.

**C.** ______ Every problem in **NP** is also in **P**

**D.** ______ **P ≠ NP**

**E.** ______ No problem is in both **P** and **NP**

**1.** True

**2.** False

**3.** False if some NP-complete problem is in P

**4.** True if some NP-complete problem is in P.

**11. Circuits (5 points).** Mark each of the circuits below as *combinational* (no feedback loops) or *sequential* (maintains state) by writing **C** or **S**, respectively, in the blanks provided.

**A.** SR flip-flop ________

**B.** multiplexer ________

**C.** memory bit ________

**D.** decoder ________

**E.** adder ________

**F.** register ________

**G.** ALU ________

**H.** program counter ________