| COS 126 | General Computer Science | Fall 2003 |
|---|---|---|

# Exam 2

This test has 13 questions worth a total of 50 points. You have 120 minutes. The exam is closed book, except that you are allowed to use a one page cheatsheet. No calculators or other electronic devices are permitted. Give your answers and show your work in the space provided. **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."*

| Problem | Score |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| Sub 1 | |

| Problem | Score |
|---|---|
| 7 | |
| 8 | |
| 9 | |
| 10 | |
| 11 | |
| 12 | |
| | |
| Sub 2 | |

| Total | |
|---|---|

**Name:**

**Login ID:**

**Precept:**

| | | |
|---|---|---|
| 1 | MF 10:00 | Paul |
| 2 | MF 11:00 | Donna |
| 3 | MF 1:30 | Shirley |
| 4 | MF 2:30 | Kevin |
| 5 | M 7:30<br>F 2:30 | Kevin |

0. **Miscellaneous. (2 points)**

   (a) Write your name and arizona login in the space provided on the front of the exam, and circle your precept number.

   (b) *Write* and sign the honor code on the front of the exam.

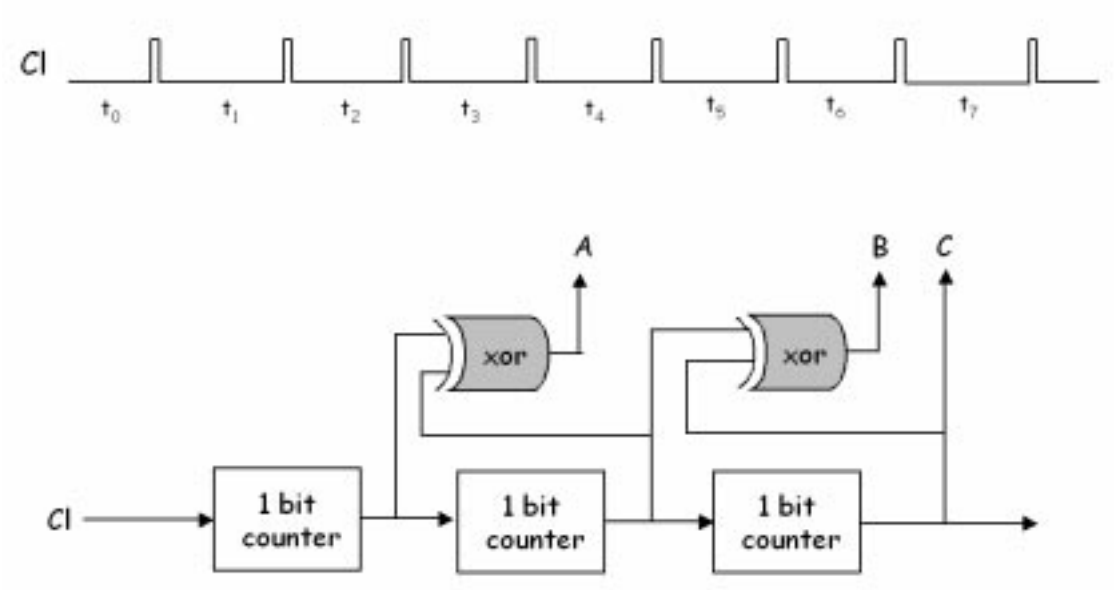
1. **Combinational circuits. (4 points)**

   Draw a combinational circuit that sorts three inputs bits. Specifically, your circuit should have three inputs ($x$, $y$, and $z$), three output ($a$, $b$, and $c$), and behave as specified in the following table. You may only using the following gates, and at most one of each type: multiway AND, multiway OR, NOT, MAJ, ODD. Recall: a majority circuit outputs true if at least half of its inputs are true, and false otherwise. An odd parity circuit returns true if an odd number of its inputs are true, and false otherwise.

| $x$ | $y$ | $z$ | $a$ | $b$ | $c$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 |

2. **Sequential circuits. (4 points)**

Consider the following circuit comprised of three 1-bit counters and two XOR gates. Recall: a 1-bit counter stores one bit, and its state changes every time its input signal goes from 1 to 0 (i.e., falling edge triggered), an XOR gate is 1 if its two inputs are different, and 0 otherwise.




(a) Assume initially that A = B = C = 0. Deduce the original values of the three 1-bit counters (listed from left to right). Circle the correct answer.

000 001 010 100

(b) The clock input to the leftmost 1-bit counter is given above. What are the values of A, B, and C at time steps $t_0, t_1, t_2, \ldots$?

i. 000, 001, 010, 011, 100, 101, 110, 111, ...

ii. 000, 011, 110, 101, 100, 111, 010, 001, ...

iii. 000, 100, 101, 100, 011, 010, 001, 111, ...

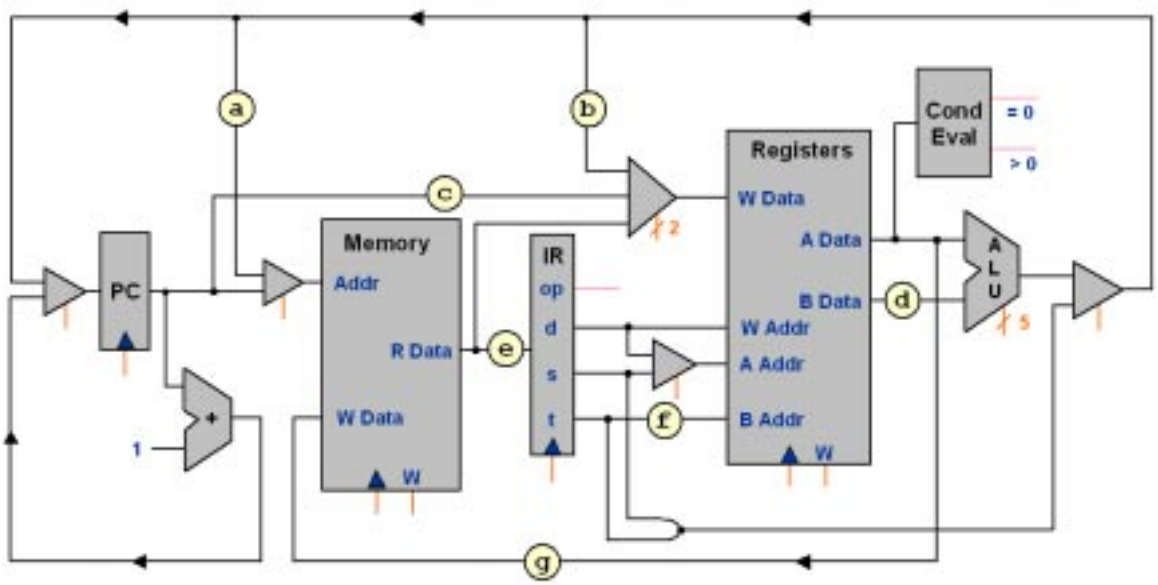
iv. 000, 100, 110, 010, 011, 111, 101, 001, ...

v. 000, 111, 110, 101, 100, 011, 010, 001, ...

3. **TOY architecture. (4 points)**

Which four of the seven connections labeled below are needed to implement an add instruction in TOY. Circle those *four* that are needed.

a. b. c. d. e. f. g.



4. **Debugging. (5 points)**

   The constructor `EditDistance(String a, String b)` is supposed to initialize the five data members, as in Assignment 8. It fails to initialize any of them as intended. Correct the constructor below.

```
public class EditDistance {
    private int M, N;
    private String x, y;
    private int[][] opt;

    public EditDistance(String a, String b) {




        M = x.length();
        N = y.length();




        int[][] opt = new int[M+1][N+1];
        for (int i = 0; i <= M; i++) {
            for (int j = 0; j <= N; j++) {
                 opt[i][j] = -1;
            }
        }







    }

}
```

5. **References. (4 points)**

What does the following program print out? Circle your answer.

```
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // multiply each coordinate by the given factor
    public void scale(int factor) {
        x = factor * x;
        y = factor * y;
    }

    public String toString() {
        return ("x = " + x + ", y = " + y);
    }

    public static void main(String[] args) {
        Point p = new Point( 5,  7);
        Point q = new Point(11, 13);
        System.out.println(p);
        System.out.println(q);

        q = p;
        p.scale(2);
        q.scale(3);
        System.out.println(p);
        System.out.println(q);
    }
}
```

6. **Abstract data types. (5 points)**

Define an ADT for a particle in the plane. Each particle has a mass, and a position (in the x and y directions). Your ADT should have appropriate data members and a constructor. It should also have a function `add(Particle a, Particle b)` that returns a `new` particle representing the *center of mass* of `a` and `b`. The center of mass of two particles with masses $m_1$ and $m_2$ and positions $(x_1, y_1)$ and $(x_2, y_2)$ has mass $m_1 + m_2$ and position $(\frac{x_1 m_1 + x_2 m_2}{m_1 + m_2}, \frac{y_1 m_1 + y_2 m_2}{m_1 + m_2})$.

```
public class Particle {









    public Particle(double mass, double x, double y) {









    }

    public static Particle add(Particle a, Particle b) {









        return new Particle(                    );
    }

}
```

7. **Linked structures. (4 points)**

The following code implements a LIFO stack ADT using a linked list. Select the proper missing code fragments.

```
public class Stack {
    private List first = null;

    private class List {
        Object item;
        List next;
        List(Object item, List next) {
            this.item = item;
            this.next = next;
        }
    }

    // is the stack empty?
    public boolean isEmpty() { return (first == null); }

    // insert a new object
    public void add(Object item) {
       first = new List(item, first);

       // MISSING FRAGMENT I

    }

    // delete and return and the most recently added object
    public Object remove() {
        Object val = first.item;
        List second = first.next;

        // MISSING FRAGMENT II

    }
}
```

___ Missing Fragment I.

___ Missing Fragment II.

A. nothing missing

B. `first.item = item;`

C. `first = item; return second;`

D. `first = second; return val;`

E. `return second.item;`

8. **Fundamental ADTs. (4 points)**

Suppose you are designing a *web crawler*. A web crawler starts from one base web page, say `http://yahoo.com`, and then examines all of its neighboring pages, and then examines all of the neighbors of the neighboring pages, etc. You plan to crawl billions of web pages, so all operations must be efficient. As a result, you consider using your COS 126 arsenal of abstract data types.

A. Graph B. Queue C. Regular expression D. Stack E. Symbol table

___ To implement a web crawler, you will need to maintain a list of web pages that you've discovered, but haven't crawled. You should examine the pages in order of increasing distance (shortest chain of hyperlinks) from `http://yahoo.com`. Which ADT would you use to maintain this list?

___ To ensure that you don't crawl a web page more than once, you need a method to determine whether or not you've already discovered the web page. Which ADT would you use to implement this?

___ When processing a web page, you need to determine all of its hyperlinks. A crude description of a hyperlink is text that begins with `http://` and consists of a sequence of non-whitespace characters other than `"`, `<` and `>`. Which ADT should you use to find such hyperlinks?

___ To study fundamental properties of the web, you want to store not just the list of web page, but also a representation of the hyperlinks. Which ADT should you use to represent the list of web pages and their pairwise connections?

9. **Hash tables. (2 points)**

Why do many programmers use hashing to implement symbol tables? Choose the best answer.

(a) To conserve memory.
(b) Because it performs well and is not hard to implement.
(c) Because the library implementation is optimized.
(d) To guarantee that lookups (get) will be faster than inserts (put).
(e) Because it supports a fast sort implementation.

10. **Regular expressions. (4 points)**

Write down a regular expression for all binary strings that either (i) contain a multiple of three 0's or (ii) end with a 0, e.g., 001101, 0100, 11110100. Circle your answer.

11. **Analysis of algorithms. (4 points)**

For each of the following algorithms, estimate the *average case* running time as a function of the input size $N$. Your answer should be 1, $N$, $N \log N$, $N^2$, $N^3$, $2^N$, or $N!$, and you may use an answer more than once.

____ Sort N randomly ordered items using insertion sort.

____ Sort N randomly ordered items using quicksort.

____ Shuffle an array of N items, using the shuffling algorithm from lecture.

____ Compute the edit distance of two strings of length N using dynamic programming, as in Assignment 7.

____ Insert or remove one item in a queue, where the queue is implemented as in lecture.

____ Insert N cities into a TSP tour using the nearest insertion heuristic, as in Assignment 6.

12. **Computational complexity. (4 points)**

For each problem on the left, put the letter of the *best* matching *guarantee* on the right. Use each answer *exactly once*.

___ Sort N real numbers.

___ Factor an N-digit integer.

___ Test whether an N-digit integer is prime.

___ Determine whether there is a winning position for black in an N-by-N game of checkers.

___ Solve a CIRCUIT-SAT problem with N gates.

___ Solve the Halting problem on programs with N characters.

A. Can be solved in $N \log N$ steps but not $N$.

B. Can be solved in a polynomial number of steps.

C. Cannot be solved in a polynomial number of steps unless P = NP.

D. Cannot be solved in a polynomial number if the RSA cryptosystem is secure.

E. Can be solved in an exponential number of steps, but not a polynomial number.

F. Cannot be solved in an exponential number of steps.