CMSC330 Fall 2010 Final Exam

Name _____

Instructions

- You have 120 minutes for to take this exam.
- This exam has a total of 180 points. An average of 40 seconds per point.
- This is a closed book exam. No notes or other aids are allowed.
- If you have a question, please raise your hand and wait for the instructor.
- Answer essay questions concisely using 1-2 sentences. Longer answers are not necessary and a penalty may be applied.
- In order to be eligible for partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score	Max Score
1	Programming languages		10
2	Ruby		6
3	Regular expressions & context free grammars		22
4	OCaml types & type inference		10
5	OCaml higher order & anonymous functions		12
6	Scoping		10
7	Parameter passing		10
8	Lazy evaluation		8
9	Garbage collection		8
10	Lambda calculus		16
11	Lambda calculus encodings		14
12	Operational semantics		16
13	Multithreading		10
14	Ruby multithreading		22
15	Markup languages		6
	Total		180

1. (10 pts) Programming languages
    a. (2 pts) Give two examples of programming language features that make it easier to write programs, but make it more difficult to detect errors in the program.

    b. (4 pts) Briefly define type safety, and describe why it is desirable.

    c. (4 pts) Briefly describe the goal of techniques such as lambda calculus and operational semantics.

2. (6 pts) Ruby
    a. (4 pts) What language feature in Ruby is similar to anonymous functions in OCaml? Explain.

    b. (2 pts) What is the output (if any) of the following Ruby program? Write FAIL if code does not execute.

```
a = { }                    # Output =
a["foo"] = 1
puts a["foo"]
puts a["bar"]
```

3. (22 pts) Regular expressions & context free grammars.
   a. (4 pts) Give a regular expression for OCaml expressions of type *int list*. Assume there are no empty lists, and all numbers are 0 or 1. Examples of strings generated by your RE include: [1], [1;0], and [0;1;1].

   b. (8 pts) Give a context free grammar for OCaml expressions of type *int list* with arbitrary levels of nesting (i.e., expressions of type *int list*, *int list list*, *int list list list*, etc.). Assume there are no empty lists, and all numbers are 0 or 1. Examples of strings generated by your CFG include: [1], [[1]], [[[1]]], [[1;0]], and [[0;1];[0]]. Lists do not need to be homogenous. For instance, [1;[1]] is allowed.

Consider the following grammar (where S = start symbol and terminals = * **0** ( )):

S → * S S | A
A → 0 | (S) | epsilon

c. (2 pts each) Indicate whether the following strings are generated by this grammar

i.	*** 0**	Yes	No	(circle one)
ii.	**(0 0)**	Yes	No	(circle one)
iii.	**0 * (0)**	Yes	No	(circle one)

d. (4 pts) Does the following prove the grammar is ambiguous? Explain.

Two derivations of the same string "*00"

S ⇒ * S S ⇒ * A S ⇒ * 0 S ⇒ * 0 A ⇒ * 0 0

S ⇒ * S S ⇒ * S A ⇒ * S 0 ⇒ * A 0 ⇒ * 0 0

4. (10 pts) OCaml Types and Type Inference

   Give the type of the following OCaml expressions:

   a. (2 pts) fun x -> x+1 Type =

   b. (3 pts) fun x -> x 1 Type =

   Write an OCaml expression with the following type:

   c. (2 pts) (int * string) list Code =

   d. (3 pts) ('a -> int) -> int Code =

5. (12 pts) OCaml higher-order & anonymous functions

   Using fold and an anonymous function, write a function *pairUp* which given an int list returns a int list list, where every pair of elements in the original list is now paired up in a list. The strings should be in the same order as the in the original list. You may assume the original list has an even number of elements (including 0). Your function must run in linear time. You may not use any library functions, with the exception of the List.rev function, which reverses a list in linear time. Solutions using recursion and/or helper functions will only receive partial credit.

```
let rec fold f a lst = match lst with
    [] -> a
    | (h::t) -> fold f (f a h) t
```

   Examples:

   pairUp [] = []
   pairUp [1;2] = [[1;2]]
   pairUp [1;2;3;4] = [[1;2];[3;4]]
   pairUp [1;2;3;4;5;6] = [[1;2];[3;4];[5;6]]

6. (10 pts) Scoping

Consider the following OCaml code.

```
let app f x = f x ;;
let proc x  = let change z = z+x in app change (x+2) ;;
(proc 3) ;;
```

a. (2 pts) What is the order the functions app, proc, and change are invoked?

b. (4 pts) What value is returned by (proc 3) with static scoping? Explain.

c. (4 pts) What value is returned by (proc 3) with dynamic scoping? Explain.

7. (10 pts) Parameter passing

Consider the following C code.

```
int i = 1;
void foo(int f, int g) {
  f  = f + g;
  g = g + 2;
}
int main( ) {
  int a[] = {3, 5, 7, 9};
  foo(i, a[i-1]);
  printf("%d %d %d %d %d\n", i, a[0], a[1], a[2], a[3]);
}
```

a. (2 pts) Give the output if C uses call-by-value

b. (4 pts) Give the output if C uses call-by-reference

c. (4 pts) Give the output if C uses call-by-name

8. (8 pts) Lazy evaluation
   a. (3 pts) Explain why lazy evaluation allows some programs to successfully execute that would not using eager evaluation.

   b. (5 pts) Rewrite the following code (using thunks) so that *foo* evaluates its argument only when it is used, even though OCaml uses call-by-value.

```
let foo f = [f] ;;
foo 1  ;;
```

9. (8 pts) Garbage collection

   Consider the following Java code.

```
class Inception {
        static DreamLayer current, up1, up2;
        private void MoviePlot( ) {
                up2 = new DreamLayer ("van");              // object 1
                up1 = new DreamLayer ("hotel");            // object 2
                current = new DreamLayer ("fortress");     // object 3
                // …dreamKick…
                current = up1;
                up1 = up2;
        }
}
```

   a. (4 pts) What object(s) are garbage when MoviePlot ( ) returns? Explain.

   b. (4 pts) List one advantage and one disadvantage of using garbage collection.

10. (16 pts) Lambda calculus

(4 pts each) Evaluate the following λ-expressions as much as possible.

a. (λx.λy.x y) a b c

b. (λx.λy.x y z) (λz.a z) b

c. (λx.λy.x y z) (λm.λn.n m o)

(2 pts each) Determine whether alpha-renaming is necessary for each of the following lambda expressions. If it is, list the variable that must be renamed.

d. (λy.λz.z a y) z x No Yes = (circle No, or give renamed var)

e. (λy.λz.y z a) a x No Yes = (circle No, or give renamed var)

11. (14 pts) Lambda calculus encodings

Consider the standard encodings for the booleans *if*, *true*, *false,* and *and*.
Using these encodings, prove that if (and true false) then x else y = y.
If you understand how they work, you do not need to expand true or false.

if a then b else c = a b c true = λx.λy.x false = λx.λy.y and = λx.λy.(xy) false

12. (16 pts) Operational semantics

a. (4 pts) In plain English, describe what the following means:

• , x :1 ; x + 2 → 3

b. (12 pts) In an empty environment, what does the expression (fun x = (fun y = y x)) z evaluate to? In other words, find a v such that you can prove the following:

• ; (fun x = (fun y = y x)) z → v

Use the operational semantics rules given in class, included here for your reference. Show the complete proof that stacks uses of these rules.

Number

$$\frac{}{\bullet; n \rightarrow n}$$

Lambda

$$\frac{}{A;\ \text{fun } x = E \rightarrow (A, \lambda x.E)}$$

Addition

$$\frac{A; E_1 \rightarrow n \qquad A; E_2 \rightarrow m}{A; + E_1 E_2 \rightarrow n + m}$$

Function application

$$\frac{A; E_1 \rightarrow (A', \lambda x.E) \qquad A; E_2 \rightarrow v \qquad A, A', x{:}v; E \rightarrow v'}{A; (E_1\ E_2) \rightarrow v'}$$

Identifier

$$\frac{}{A; x \rightarrow A(x)}$$

13. (10 pts) Multithreading

Consider the following attempt to implement producer/consumer pattern w/ Java 1.4.

```
class Buffer {
    Buffer ( ) {
      Object buf = null;
      boolean empty = true;
    }
    void produce(o) {
      synchronize (buf) {
1.      if (!empty) wait( );
2.      empty = false;
3.      notifyAll( );
4.      buf = o;
       }
    }
```

```
    Object consume( ) {
      synchronize (buf) {
5.      if (empty) wait( );
6.      empty = true;
7.      notifyAll( );
8.      return buf;       // also releases lock
       }
    }
}
t1 = Thread.run { produce(1); }
t2 = Thread.run { produce(2); }
t3 = Thread.run { x = consume( ); }
t4 = Thread.run { y = consume( ); }
t5 = Thread.run { z = consume( ); }
t6 = Thread.run { produce(3); }
```

In the following, give schedules as a list of thread name/line number/range pairs, e.g., (t1, 1-4), (t2, 1), (t3, 5-8). For instance, one schedule under which x=1 and y=2 is (t1, 1-4), (t3, 5-8), (t2, 1-4), (t4, 5-8)

a. (2 pts) Give a schedule under which x = 2 and y = 1.

b. (4 pts) Give a schedule under which x = 2 and y = 2, or argue that no such schedule is possible.

c. (4 pts) Give a schedule under which x = 1 and thread 4 blocks, or argue that no such schedule is possible.

14. (22 pts) Ruby multithreading

Using Ruby monitors and condition variables, write a Ruby function simulate(M,N) that implements the following simulation of a dance club. M girls and N guys arrive at a club. Each guy is assigned a number between 0 and N-1, and each girl is assigned a number between 0 and M-1. Once at the club, each girl dances 10 times, each time picking any guy who is not currently dancing with another girl. Each dance lasts 0.01 seconds in real time (i.e., call sleep 0.01). Print out a message "X dancing with Y" for girl X and guy Y at the start of each dance. The action for each girl must be executed in a separate thread. You must allow multiple couples to dance at the same time (i.e., while calling sleep 0.01). Once all girls have finished dancing, the simulation is complete.

You must use monitors to ensure there are no data races, and condition variables to ensure girls efficiently wait if all guys are currently dancing. You may only use the following library functions.

Allowed functions:

```
n.times { |i| … }        // executes code block n times, with i = 0…n-1
a = [ ]                  // returns new empty array
a.empty?                 // returns true if array a is empty
a.push(x)                // pushes (adds) x to array a
x = a.pop                // pops (removes) element of a and assigns it to x
a.each { |x| … }         // calls code block once for each element x in a
m = Monitor.new          // returns new monitor
m.synchronize { … }      // only 1 thread can execute code block at a time
c = m.new_cond           // returns conditional variable for monitor
c.wait_while { … }       // sleeps while code in condition block is true
c.wait_until { … }       // sleeps until code in condition block is true
c.broadcast              // wakes up all threads sleeping on condition var c
t = Thread.new {… }      // creates thread, executes code block in new thread
t.join                   // waits until thread t exits
```

15. (6 pts) Markup languages

Creating your own XML tags, write an XML document that organizes the following information about turtles. Sea turtles live in the ocean and can grow to 2000 pounds. Tortoises live on land and can grow to 660 pounds. Terrapins live in rivers and can grow to 130 pounds.