CMSC330 Fall 2010 Final Exam Solutions

1. (8 pts) Programming languages
   a. (4 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.
      Implicit declarations, dynamic types, weak typing, etc…

   b. (4 pts) Briefly define type safety, and describe why it is desirable.
      Ensuring operations performed on a value are those appropriate for the type of the value. Desirable since it reduces chance of program errors.

   c. (4 pts) Briefly describe the goal of techniques such as lambda calculus and operational semantics.
      Determining the meaning (semantics) of a program, in order to prove its correctness or safety.

2. (6 pts) Ruby
   a. (4 pts) What language feature in Ruby is similar to anonymous functions in OCaml? Explain.
      Code blocks, since they define unnamed functions that can be passed to methods as arguments.

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

```
a = { }                        # Output = 1 nil
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].
      [(0|1)(;(0|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 need to be homogenous. For instance, [1;[1]] is allowed.
      S → [L]
      L → N ; L | N
      N → 0 | 1 | S

   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"

      $S \Rightarrow * S S \Rightarrow * A S \Rightarrow * 0 S \Rightarrow * 0 A \Rightarrow * 0 0$

      $S \Rightarrow * S S \Rightarrow * S A \Rightarrow * S 0 \Rightarrow * A 0 \Rightarrow * 0 0$

      No, proof requires multiple parse trees, multiple left-most derivations, or multiple right-most derivations.

4. (10 pts) OCaml Types and Type Inference

   Give the type of the following OCaml expressions:
   a. (2 pts) fun x -> x+1 **Type = int -> int**
   b. (3 pts) fun x -> x 1 **Type = (int -> 'a) -> 'a**

   Write an OCaml expression with the following type:
   c. (2 pts) (int * string) list **Code = [1, "foo"] or [(1,"foo")]**
   d. (3 pts) ('a -> int) -> int **Code = fun x -> 1+(x y)**

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 may 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]]
```

let pairUp l = List.rev (fold (fun a h -> match a with
[] -> [[h]] **// empty a = 1st elem**
| [x]::t -> [x;h]::t **// a begins w/ 1 elem list, add to list**
| [x;y]::t -> [h]::a **// a begins w/ 2 elem list, make new elem**
) [] l) ;; **// initial a = []**

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?

Proc, app, change.

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

8, since the value of x in change is the argument to proc (3).

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

10, since the value of x in change is the argument to app (5).

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

1 3 5 7 9 since i & a are unchanged by calling foo.

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

4 5 5 7 9 since i & a[0] are changed (to i+a[0] and a[0]+2) by calling foo.

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

4 3 5 7 11 since i is changed (to i+a[0]) and a[3] is changed (to a[3]+2) by calling foo.

8. (8 pts) Lazy evaluation
   a. (3 pts) Explain why lazy evaluation allows some programs to successfully execute that would not using eager evaluation.
      Unused function parameters that would cause errors when evaluated.

   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  ;;
```

```
let foo f = [f ()]
foo (fun () -> 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.
      Object 3 (DreamLayer (“fortress”)), since it can no longer be accessed.

   b. (4 pts) List one advantage and one disadvantage of using garbage collection.
      Advantages = less programmer effort, fewer memory errors
      Disadvantages = more memory use, extra work during 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 → **(λy.a y) b c → a b c**
   b. (λx.λy.x y z) (λz.a z) b → **(λy.(λz.a z) y z) b → (λz.a z) b z → a b z**
   c. (λx.λy.x y z) (λm.λn.n m o) → **λy.(λm.λn.n m o) y z → λy.(λn.n y o) z**
   → λy.z y 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 = z**   (circle No, or give renamed var)
   e. (λy.λz.y z a) a x   **No**   Yes =   (circle No, or give renamed var)

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

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 (and true false) then x else y → **(and true false) x y**
→ **(λx.λy.(xy) false) true false x y**
→ **(λy.(true y) false) false x y**
→ **true false false x y**
→ **false x y**
→ **y**

12. (16 pts) Operational semantics
a. (4 pts) In plain English, describe what the following means:
$\bullet$, x :1 ; x + 2 ⟶ 3
In the environment created by binding x to 1, evaluating x+2 yields 3.

b. (12 pts) In an empty environment, what the expression (fun x = (fun y = y x)) 1 evaluate to? In other words, find a v such that you can prove the following:
$\bullet$; (fun x = (fun y = y x)) 2 ⟶ 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
$$\frac{}{A;\ x \rightarrow A(x)}$$

• ; (fun x = (fun y = y x)) ⟶ (•, λx.(fun y = y x)) **// value of closure**
• ; 2 ⟶ 2 **// value of argument**
•, x:2 ; (fun y = y x) ⟶ (x :2, (λy.y x)) **// value of closure body**
// evaluated in new env

$$\frac{}{\textbf{• ; (fun x = (fun y = y x)) 2} \longrightarrow \textbf{(x:2, λy.y x))}}$$

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). Threads whose execution do not affect the result may be ignored.

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

   (t2, 1-4), (t3,5-8), (t1,1-4), (t4,5-8) etc…

   Have t3 execute after t2, and t4 execute after t1

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

   (t4,5), (t2, 1-4), (t3,5-8), (t4,6-8) OR

   (t3,5), (t2, 1-4), (t4,5-8), (t3,6-8) etc…

   Have one consumer thread (either t3 or t4) misbehave by waiting on 5, then returning and continuing execution even though condition is not valid (i.e., empty = true), causing it to read value 2 already read by other consumer

c. (4 pts) Give a schedule under which x = 1 and thread 4 blocks.

   (t1,1-4), (t2, 1), (t6, 1), (t3,5-8), (t2,2-4), (t6,2-4), (t5,5-8), (t4,5) etc…

   Have producer thread t2 & t6 misbehave by waiting on 1, then returning and continuing execution even though condition is not valid (i.e., empty = false), for both, causing one producer to overwrite buf value already produced by other producer thread. One consumer will then hang because there are insufficient producers.

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 from 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
a.each { |x| … }	// calls code block once for each element x in a
x = a.pop	// pops (removes) element of a and assigns it to x
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

```
require "monitor"
Thread.abort_on_exception = true   # to avoid hiding errors in threads
class DanceHall
  def initialize
    @m = Monitor.new
    @c = @m.new_cond
    @num = 1;
    @guys = []
  end
  def goGirl
    me = 0
    g = 0
    @m.synchronize {
       me = @num
       @num = @num+1
    }
    10.times {
      @m.synchronize {
        @c.wait_while { @guys.empty? }
        g = @guys.pop
        puts "#{me} dancing with #{g}"
        $stdout.flush
      }
      sleep 0.01
      @m.synchronize {
        @guys.push(g)
        @c.broadcast
      }
    }
  end
  def simulate(numGirls,numGuys)
    numGuys.times { |i|
      @guys.push(i)
    }
    threads = []
    numGirls.times { |i|
      t = Thread.new { goGirl }
      threads.push(t)
      }
    threads.each { |t| t.join }
    puts "All done!"
  end
end

d = DanceHall.new
d.simulate(10,3)
```

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.

```
<turtles>
    <turtle type>
        <name>Sea turtle</name>
        <pounds>2000</pounds >
        <habitat>ocean</habitat >
    </turtle type >
    <turtle type>
        <name> Tortoise </name>
        <pounds>660</pounds >
        <habitat>land</habitat >
    </turtle type >
    <turtle type>
        <name>Sea Terrapin </name>
        <pounds>130</pounds >
        <habitat>rivers</habitat >
    </turtle type >
</turtles >
```