CMSC330 Fall 2011 Final Exam Grading Key

1. (12 pts) Programming languages

   a. (3 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 the chance of program errors.

   b. (3 pts) Briefly describe the goal of techniques such as lambda calculus and operational semantics.
      Clearly describe the effect of evaluating a piece of code.

   c. (3 pts) Briefly define polymorphism, and explain why it can be a useful feature for a programming language.
      Allow different data types to be handled using the same interface. Useful because allows code/interface to be reused.

   d. (3 pts) Briefly describe how syntax & semantics apply to markup & query languages such as XML and SQL, even though they are not full programming languages.
      Syntax & semantics are still needed to define how the language looks, and what it does.

2. (10 pts) Ruby

   Consider the following code:

```
a[2] = 3
x = a.include?(2)
y = a.include?(3)
```

   a. (2 pts) If the code is preceded by the line a = [ ], what are the values of x & y?
      x = false, y = true

   b. (2 pts) If the code is preceded by the line a = { }, what are the values of x & y?
      x = true, y = false

   c. (3 pts) What is another important difference between calling a.include?( ) when "a" is an Array vs. a Hash? Hint, it's not whether the answer is true or false.
      Efficency: operation is linear time for Array, constant time for Hash.

   d. (3 pts) What is a simpler alternative to writing a.include?( ) when "a" is a Hash?
      Use "if a[x] != nil" instead of "a.include?(x)"

3. (14 pts) Context free grammars and parsing

Consider the following grammar

S →	ABc	|	Bd
A →	a	|	cB
B →	b	|	ɛpsilon

a. (2 pts) Give a leftmost derivation of the string "ac"

S → ABc → aBc → ac

b. (6 pts) Calculate FIRST sets for S, A, B

FIRST(S) = { a, c, b, d }
FIRST(A) = { a, c }
FIRST(B) = { b, ε }

c. (6 pts) Using pseudocode, write *only* the parse_A function found in a recursive descent parser for the grammar. You may assume the functions parse_S , parse_B already exist.

Use the following utilities:

lookahead	Variable holding next terminal
match (x)	Function to match next terminal to x
error ( )	Reports parse error for input

parse_A( ) { // your code starts here

```
parse_A( ) {
        if (lookahead == "a") {        // A → a
                match("a");
        }
        else if (lookahead == "c") {  // A → cB
                match("c");
                parse_B( );
        }
        else error( );
}
```

4. (10 pts) OCaml Types and Type Inference

Give the type of the following OCaml expressions:

a. (2 pts) fun x -> x 2 4 **Type = (int -> int -> 'a) -> 'a**
b. (3 pts) fun x -> fun y-> 3 **Type = 'a -> 'b -> int**

Write an OCaml expression with the following type:

c. (2 pts) (int * string list) **Code = (2,["foo"])**
d. (3 pts) 'a -> int -> int **Code = fun x -> fun y -> y+2**
= fun x y -> y+2

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

Using fold and an anonymous function, write a function *getSeconds* which given an 'a list list returns 'a list, composed of the 1st elements of every list in the original list. The elements should be in the same order as the in the original list. You may assume there all lists in the original list have at least 2 members. 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:
getSeconds [] = []
getSeconds [[1;2]] = [2]
getSeconds [[1;2];[3;4]] = [2;4]
getSeconds [[1;2];[3;4];[5;6;7]] = [2;4;6]

```
let getSeconds l = List.rev (fold (fun a h -> match h with
      | _::y::_ -> y::a          // prepend 2nd elem of h to a
      | [x;y;_] -> y::a          // prepend 2nd elem of h to a
      ) [ ] 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+5) ;;
(proc 3) ;;
```

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

Proc, app, change.

Value computed by proc is (x+5)+x, where the 1st x is the argument to change, and the 2nd x is the value bound to x in change

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

11 , since the value of x in change is the formal parameter x in proc (3). I.e., (3+5)+3

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

16, since the value of x in change is the formal parameter x in app (8). I.e., (3+5)+8

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 execute 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. (14 pts) Lambda calculus

    Evaluate the following λ-expressions as much as possible.

    a. (3 pts) (λx.λy.x y) y z x → **(λx.λa.x a) y z x → (λa.y a) z x → y z x**
    b. (3 pts) (λx.λy.y x) a (λz.b z) c → **(λy.y a) (λz.b z) c → (λz.b z) a c → b a c**

Lambda calculus encodings

c. (8 pts) Using encodings, show 1*3 =>* 3. Show each beta-reduction.
=>* indicates 0 or more steps of beta-reduction

M * N = λx.(M (N x))
1 = λf.λy.f y
2 = λf.λy.f (f y)
3 = λf.λy.f (f (f y))
4 = λf.λy.f (f (f (f y)))

1*3 => **λx.(1 (3 x))**
=> λx.(1 (λf.λy.f (f (f y)) x))
=> λx.(1 (λy.x (x (x y))))
=> λx.((λf.λy.f y) (λy.x (x (x y))))
=> λx.(λy.(λy.x (x (x y))) y)
=> λx.(λy.x (x(x y)))
=> 3

OR

1*3 => **λx.(1 (3 x))**
=> λx.((λf.λy.f y) (3 x))
=> λx.(λy.(3 x) y)
=> λx.(λy.((λf.λy.f (f (f y))) x) y)
=> λx.(λy.(λy.x (x(x y))) y)
=> λx.(λy.x (x(x y)))
=> 3

(12 pts) Operational semantics

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

y:1 ; fun x = x → (y:1, λx.x)

In the environment created by binding y to 1, evaluating fun x = x yields the closure with environment y:1 and the code λx.x (identity function)

e. (8 pts) In an empty environment, what does the expression (fun x = x+2) evaluate to? In other words, find a v such that you can prove the following:

• ; (fun x = 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.

• ; (fun x = x+2) → (•, λx.x+2)) **// value of closure**

Number

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

Addition

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

Identifier

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

Lambda

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

Function application

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

11. (14 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( ); }
```

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.

(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. (8 pts) Explain why the given Java code allows data races and why deadlock may occur.

Because wait is not called in a while loop. Multiple threads may be woken, so the condition may not be true by the time a thread wakes up.

Deadlock:
(t1,1-4), (t2, 1), (t6, 1), (t3,5-8), (t2,2-4), (t6,2-4), (t5,5-8), (t4,5) etc…
Deadlock can occur by having 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.

12. (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
```

```
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)
```

13. (8 pts) Polymorphism

a. (4 pts) Does OCaml uses ad hoc or parametric polymorphism? Briefly explain.

OCaml uses parametric polymorphism, since the same identical code handles parameters of different types.

b. (4 pts) Briefly explain why "? extends A" is needed for Java generics.

Wildcards such as ? are needed to support subtyping for containers, and "extends A" is needed to restrict what classes can be matched by the wildcard.

14. (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 >
```