CMSC330 Fall 2013 Final Exam

Name: _____

Discussion Time	**9am**	**10am**	**11am**	**Noon**	**1pm**
TA Name (circle):	**Ilse**	**Daniel**	**Casey**	**Yoav**	**Ilse**
	Richard	**Richard**	**Richard**		

Instructions

- Do not start this test until you are told to do so!
- You have 120 minutes to take this exam.
- This exam has a total of 200 points, so allocate 36 seconds for each point.
- This is a closed book exam. No notes or other aids are allowed.
- Answer essay questions concisely in 2-3 sentences. Longer answers are not needed.
- For partial credit, show all of your work and clearly indicate your answers.
- Write neatly. Credit cannot be given for illegible answers.

	Problem	Score
1	Programming Languages	/20
2	Ruby & OCaml	/16
3	Scoping	/9
4	Parameter Passing	/12
5	Lazy Evaluation	/10
6	Garbage Collection	/8
7	Polymorphism	/12
8	Lambda Calculus	/20
9	Multithreading	/15
10	Ruby Multithreading	/30
11	Prolog	/20
12	Prolog Programming	/28
	Total	/200

HONOR PLEDGE: I pledge on my honor that
I have not given or received any unauthorized
assistance on this assignment/ examination. SIGNATURE: _____

1. (20 pts) Programming languages

    a. (4 pts) Explain why programming languages use types (e.g., int x, char y).

    b. (4 pts) Explain why static types do not ensure a strong type system.

    c. (4 pts) Explain why programming languages use scopes (e.g., {…}, begin … end).

    d. (4 pts) Explain when programming languages need to use closures.

    e. (4 pts) Explain why closures are similar to objects.

2. (16 pts) Ruby & OCaml
What is the output (if any) of the following Ruby programs? Write FAIL if code does not execute.

a. (2 pts)
```
a = “Captain America”
if a =~ /([a-z]+)/ then
   puts $1
end
```

b. (2 pts)
```
a = { }
a[“Iron”] = “Man”
puts a[“Iron”]
```

c. (4 pts) Give the type of the following OCaml expression
```
fun x -> fun y -> y::x
```

d. (4 pts) Write an OCaml expression with the following type
```
(‘a -> ‘a list) -> int list
```

e. (4 pts) Give the value of the following OCaml expression. If an error exists, describe the error.

```
fold (fun x -> fun y -> y::x) [ ] [2;3;4]
```

3. (9 pts) Scoping

Consider the following OCaml code.

```
let app f y = let x = 4 in (f y)+1 ;;
let proc x  = let change z = z-x in app change (x+3) ;;
(proc 2) ;;
```

a. (3 pts) What value is returned by (proc 2) with static scoping? Explain.

b. (6 pts) What value is returned by (proc 2) with dynamic scoping? Explain.

4. (12 pts) Parameter passing

Consider the following C code.

```
int i = 2;
void foo(int f, int g) {
  f  = f + g;
  g = g + 4;
}
int main( ) {
  int a[] = {1, 1, 1, 1};
  foo(i, a[i-2]);
  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. (5 pts) Give the output if C uses call-by-reference

c. (5 pts) Give the output if C uses call-by-name

5. (10 pts) Lazy evaluation

Rewrite the following OCaml code using thunks so that foo uses lazy evaluation.

```
let foo x = x + 1;;
foo (2+3);;
```

6. (8 pts) Garbage collection

Consider the following Java code.

```
class AvengersMovie {
        static Avenger x, y, z;
        private void MoviePlot( ) {
                x = new Avenger ("Iron Man");          // object 1
                y = new Avenger ("Black Widow");       // object 2
                z = new Avenger ("Bruce Banner");      // object 3
                // transformation!
                z = new Avenger("Hulk");               // object 4
        }
}
```

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

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

7. (12 pts) Polymorphism
    a. (4 pts) Explain why + in Java is an example of ad hoc polymorphism.

    b. (4 pts) Briefly explain why Java generics is an example of parametric polymorphism.

    c. (4 pts) Briefly explain why "?" is used for Java generics, such as Set<?>.

8. (20 pts) Lambda calculus

Evaluate the following λ-expressions as much as possible.

a. (4 pts) (λx.λy.y x y x) y x z

b. (6 pts) (λx.λy.λz.y z x) (λc.c) (λa.λb.b a) d

Lambda calculus encodings

c. (10 pts) Using encodings, show succ 3 =>* 4. Show each beta-reduction.

=>* indicates 0 or more steps of beta-reduction

succ = λz.λf.λy.f (z f y)
0 = λf.λy.y
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)))
5 = λf.λy.f (f (f (f (f y))))

9. (15 pts) Multithreading

```
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;
    }
  }
}
```

Consider the multithreaded Java 1.4 code above. Assume there are multiple producer & consumer threads being executed in the program, but just a *single* Buffer object. If we freeze program execution at some point in time, each thread *T* will have just finished executing some statement *S* (i.e., statement *S* will have been the last statement executed by thread *T*). We wish to examine the state of these threads.

a. (4 pts) Is it possible given two threads x and y for the last statement executed by thread x to be statement 2 and thread y to be statement 7 in the code above? Explain your answer.

b. (5 pts) Is it possible given two threads x and y for the last statement executed by thread x to be statement 5 and thread y to be statement 3 in the code above? Explain your answer.

c. (6 pts) Is it possible in the code above for two threads calling consume( ) to get the same value from the Buffer object? Explain your answer.

10. (30 pts) Ruby multithreading

Using Ruby monitors and condition variables, write a Ruby function simulate(M,N) that implements the following simulation of a superhero training facility with M heroes and N training rooms. Each hero arrives and is assigned a number between 0 and M-1, and each training room is assigned a number between 0 and N-1.

Once at the facility, each hero picks a room and trains with another superhero, repeating 10 times. Each room holds up to 2 heroes at a time. Training sessions occur with pairs of superheroes and lasts 0.01 seconds (i.e., call sleep 0.01). Print out a message "Room Z training X & Y" for heroes X and Y training together in each training session in room Z. The choice of a room is not significant. You may use a *getRoom*() function that returns a random room between 0 and N-1 to assign heroes to a room.

Each superhero must be implemented in a separate thread. You must allow pairs of heroes to train at the same time in different rooms (i.e., while one pair is calling sleep 0.01). Once all heroes have finished training, the simulation is complete. You may assume the simulation will automatically be terminated if all active superheroes are waiting in rooms.

You must use monitors to ensure there are no data races, and condition variables to ensure heroes efficiently wait if training rooms are all occupied. The 1st hero entering a training room will also need to efficiently wait for a 2nd hero to join the room (i.e., should wait and only wake up when the 2nd hero enters room). 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.length                 // returns size of array
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
```

Hint: You can start by modifying the following sequential version of the simulation:

```
def goTrain(me)
   10.times {
      r = getRoom()
        if $rooms[r].empty?
          $rooms[r].push(me)
        else
          h = $rooms[r].pop
          sleep 0.01  # train!
          puts "Room #{r} training #{me} & #{h}"
        end
      } } end
```

```
def simulate(m,n)
  $nHeroes = m;
  $nRooms = n;
  $rooms = []

  # prepare training rooms
  $nRooms.times { |i| $rooms[i] = [] }

  # start hero training
  $nHeroes.times { |me| goTrain(me) }
end
```

11. (20 pts) Prolog

Given the following clauses, list all answers returned by the following queries.

```
avenger(ironman).
avenger(thor).
sibling(thor,loki).
asgardian(thor).
asgardian(X) :- sibling(Y,X),asgardian(Y).
train1(X,Y) :- avenger(X),!,avenger(Y).
train2(X,Y) :- avenger(X),X\=Y,avenger(Y).
```

a. (2 pts) ?- avenger(thor).

b. (2 pts) ?- asgardian(A).

c. (2 pts) ?- train1(A,B).

d. (3 pts) ?- train2(A,thor).

e. (3 pts) ?- train2(thor,A).

```
foo([H,H|T],H).
foo([_|T],R) :- foo(T,R).
```

f. (2 pts) ?- foo([1,2],A).

g. (3 pts) ?- foo([2,2],A).

h. (3 pts) ?- foo([1,2,2,3,4,4],A)

12. (28 pts) Prolog programming

Write a prolog function findDups(A,X,Y) that given a list A, looks for adjacent duplicate values X starting at index Y, where the first element has index 0. If there are multiple duplicates, backtracking should return additional answers, starting from left to right in the list. findDups( ) fails if there are no duplicates in A. You may use the operators =, \=, \+, is, +, -, [H|T], and !. You do not need to worry about efficiency.

Examples:

?- findDups([],X,Y). false. ?- findDups([1],X,Y). false.	?- findDups([1,2],X,Y). false. ?- findDups([1,1],X,Y). X=1, Y=0.	?- findDups([0,1,1,2,2],X,Y). X=1, Y=1; X=2, Y=3.