CMSC330 Spring 2009 Final Exam

Name ________________________________

Discussion Time (circle one): 9am 10am

Do not start this exam until you are told to do so!

Instructions

- You have 120 minutes for to take this midterm.
- This exam has a total of 160 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 2-3 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		14
2	Regular expressions & CFGs		8
3	Finite automata		10
4	Parsing		12
5	OCaml types & type inference		12
6	OCaml programming		10
7	Scoping		8
8	Polymorphism		9
9	Multithreading		20
10	Lambda calculus		16
11	Lambda calculus encodings		16
12	Operational semantics		8
13	Markup languages		8
14	Garbage collection		9
	Total		160

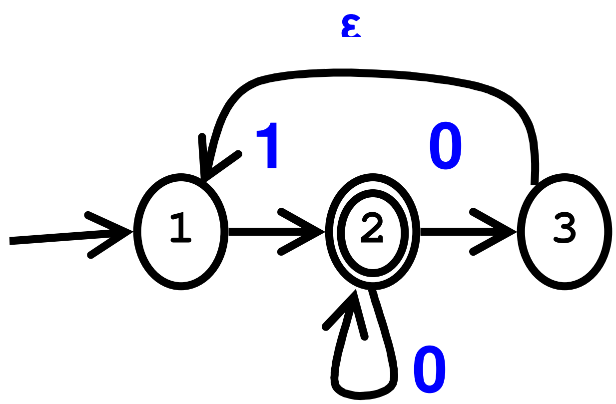
1. (14 pts) Programming languages
    a. (6 pts) List 3 different design choices for *parameter passing* in a programming language. Which choice is seldom used in modern programming languages? Explain why.
    b. (4 pts) List 2 different design choices for *type declarations* in a programming language. Which choice is seldom used in modern programming languages? Explain why.
    c. (4 pts) List 2 different design choices for determining *scoping* in a programming language. Which choice is seldom used in modern programming languages? Explain why.

2. (8 pts) Regular expressions and context free grammars

    Give a
    a. (4 pts) Regular expression for binary numbers with an even number of 1s.
    b. (4 pts) Context free grammar for binary numbers with twice as many 1s as 0s

3. (10 pts) Finite automata

    Apply the subset construction algorithm to convert the following NFA to a DFA.
    Show the NFA states associated with each state in your DFA.



4. (12 pts) Parsing

    Consider the following grammar:

    S → Ac | a
    A → bS | epsilon

    a. (6 pts) Compute First sets for S and A
    b. (6 pts) Write the parse_A( ) function for a predictive, recursive descent parser for the grammar (You may assume parse_S( ) has already been written, and match( ) is provided).

5. (12 pts) OCaml Types and Type Inference
   a. (4 pts) Give the type of the following OCaml expression
      let f x y z = y (x z)   **Type =**
   b. (6 pts) Write an OCaml expression with the following type
      int -> (int * int -> 'a) -> 'a   **Code =**
   c. (2 pts) Give the value of the following OCaml expression. If an error exists, describe the error.
      let x y = x in 3   **Value =**

6. (10 pts) OCaml Programming

   Consider the OCaml type *bst* implementing a binary tree:

```
type tree =
   Empty
   | Node of int * tree * tree;;

let rec equal = …                 (* type =  (tree * tree) -> bool *)
```

   Implement a function *equal* that takes a tuple argument (t1, t2) that returns true if the two trees t1 and t2 are of the same shape *and* equivalent nodes in the trees have the same value, else returns false.

7. (8 pts) Scoping

   Consider the following OCaml code.

```
let app f y = let x = 5 in let y = 7 in let a = 9 in f y ;;
let add x y = let incr a = a+y in app incr x ;;
 (add 1 (add 2 3)) ;;
```

   a. (2 pts) What value is returned by (add 1 (add 2 3)) with static scoping? Explain.
   b. (6 pts) What value is returned by (add 1 (add 2 3)) with dynamic scoping? Explain.

8. (9 pts) Polymorphism
   Consider the following Java classes:

```
class A { public void a( ) { … } }
class B extends A { public void b( ) { … } }
class C extends B { public void c( ) { … }}
```

   (3 pts each) Explain why the following code is or is not legal
   a. int count(Set<B> s) { … } … count(new TreeSet<C>( ));
   b. int count(Set<? extends B> s) { … } … count(new TreeSet<C>());
   c. int count(Set<? super C> s) { for (A x : s) x.a( ); … }

9. (20 pts) Multithreading

Using Ruby monitors and condition variables, you must implement a multithreaded simulation of factories producing chopsticks for philosophers. Factories continue to *produce* chopsticks one at a time, placing them in a single shared market. The market can only hold 10 chopsticks at a time. Philosophers enter the market to *acquire* 2 chopsticks.

Helpful functions:

```
m = Monitor.new        // returns 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.broadcast            // wakes up all threads sleeping on condition var
t = Thread.new {… }    // creates thread, executes code block in new thread
t.join                 // waits until thread t exits
```

a. (14 pts) Implement a thread-safe class Market with methods initialize, produce, and acquire that can support multiple multi-threaded factories and philosophers.

```
require "monitor.rb"
class Market
  def initialize
       # initialize synchronization, number of chopsticks
  end
  def produce
       # produces 1 chopstick if market is not full ( < 10 )
       # increases number of chopsticks in market by 1
  end
  def acquire
       # acquires 2 chopsticks if market has 2 or more chopsticks
       # decreases number of chopsticks in market by 2
  end
end
```

b. (6 pts) Write a simulation with 2 factories and 2 philosophers using the market. Each factory and philosopher should be in a separate thread. The simulation should exit *after* both philosophers acquire a pair of chopsticks.

10. (16 pts) Lambda calculus

(4 pts) Find all free (unbound) variables in the following λ-expressions

a. (λa. c b) λb. a

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

b. (λx.λy.y x) a b

c. (λz.z x) (λy.y x)

d. (4 pts) Write a small λ-expression which requires alpha-conversion to evaluate properly.

11. (16 pts) Lambda calculus encodings

Prove the following using the appropriate λ-calculus encodings, given**:**

$1 = \lambda f.\lambda y.f\ y$
$2 = \lambda f.\lambda y.f\ (f\ y)$
$3 = \lambda f.\lambda y.f\ (f\ (f\ y))$
$4 = \lambda f.\lambda y.f\ (f\ (f\ (f\ y)))$
$M * N = \lambda x.(M\ (N\ x))$
$Y = \lambda f.(\lambda x.f\ (x\ x))\ (\lambda x.f\ (x\ x))$
$succ = \lambda z.\lambda f.\lambda y.f\ (z\ f\ y)$

a. (10 pts) $2 * 2 = 4$
b. (6 pts) (Y succ) x = succ (Y succ) x // you do not need to expand succ

12. (8 pts) Operational semantics

Use operational semantics to determine the values of the following OCaml codes:

(fun x = + 4 x) 2

13. (8 pts) Markup languages

Creating your own XML tags, write an XML document that organizes the following information: Yoda is a 900 year old Jedi with rank Grandmaster, Obi-Wan is a 36 year old Jedi with rank Master, Anakin is an 9 year old Jedi with rank Padawan.

14. (9 pts) Garbage collection


Consider the following Java code.

```
Jedi Darth, Anakin;
private void plotTwist( ) {
        Anakin = new Jedi( );  // object 1
        Darth = new Jedi( );     // object 2
        Darth = Anakin;
        Anakin = Darth;
}
```

a. (3 pts) What object(s) are garbage when plotTwist ( ) returns? Explain why.
b. (3pts) Explain why stop-and-copy has to copy live objects.
c. (3 pts) How can garbage collection take advantage of the fact an object is from an older generation?