Name:____

CSE 505, Fall 2007, Final Examination
10 December 2007

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 12:20.**
- You can rip apart the pages, but please write your name on each page.
- There are **96 points** total, distributed *unevenly* among **5** questions.
- Most questions have multiple parts and the last question is much shorter and worth fewer points.

Advice:

- Read questions carefully. Understand a question before you start writing.
- Write down thoughts and intermediate steps so you can get partial credit.
- The questions are not necessarily in order of difficulty. **Skip around.** In particular, make sure you get to all the problems.
- If you have questions, ask.
- Relax. You are here to learn.

Name:____

For your reference (page 1 of 2):

$$\begin{array}{rcl} e & ::= & \lambda x.\ e \mid x \mid e\ e \mid c \mid \{l_1 = e_1, \ldots, l_n = e_n\} \mid e.l_i \mid \mathsf{fix}\ e \\ v & ::= & \lambda x.\ e \mid c \mid \{l_1 = v_1, \ldots, l_n = v_n\} \\ \tau & ::= & \mathsf{int} \mid \tau \to \tau \mid \{l_1 : \tau_1, \ldots, l_n : \tau_n\} \end{array}$$

$\boxed{e \to e' \text{ and } \Gamma \vdash e : \tau \text{ and } \tau_1 \leq \tau_2}$

$$\frac{}{(\lambda x.\ e)\ v \to e[v/x]} \qquad \frac{e_1 \to e_1'}{e_1\ e_2 \to e_1'\ e_2} \qquad \frac{e_2 \to e_2'}{v\ e_2 \to v\ e_2'} \qquad \frac{e \to e'}{\mathsf{fix}\ e \to \mathsf{fix}\ e'} \qquad \frac{}{\mathsf{fix}\ \lambda x.\ e \to e[\mathsf{fix}\ \lambda x.\ e/x]}$$

$$\frac{}{\{l_1 = v_1, \ldots, l_n = v_n\}.l_i \to v_i}$$

$$\frac{e_i \to e_i'}{\{l_1 = v_1, \ldots, l_{i-1} = v_{i-1}, l_i = e_i, \ldots, l_n = e_n\} \to \{l_1 = v_1, \ldots, l_{i-1} = v_{i-1}, l_i = e_i', \ldots, l_n = e_n\}}$$

$$\frac{}{\Gamma \vdash c : \mathsf{int}} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)} \qquad \frac{\Gamma, x : \tau_1 \vdash e : \tau_2}{\Gamma \vdash \lambda x.\ e : \tau_1 \to \tau_2} \qquad \frac{\Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash e_1\ e_2 : \tau_1} \qquad \frac{\Gamma \vdash e : \tau \to \tau}{\Gamma \vdash \mathsf{fix}\ e : \tau}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \ldots \qquad \Gamma \vdash e_n : \tau_n \qquad \text{labels distinct}}{\Gamma \vdash \{l_1 = e_1, \ldots, l_n = e_n\} : \{l_1 : \tau_1, \ldots, l_n : \tau_n\}} \qquad \frac{\Gamma \vdash e : \{l_1 : \tau_1, \ldots, l_n : \tau_n\} \qquad 1 \leq i \leq n}{\Gamma \vdash e.l_i : \tau_i}$$

$$\frac{\Gamma \vdash e : \tau \qquad \tau \leq \tau'}{\Gamma \vdash e : \tau'}$$

$$\frac{}{\{l_1{:}\tau_1, \ldots, l_n{:}\tau_n, l{:}\tau\} \leq \{l_1{:}\tau_1, \ldots, l_n{:}\tau_n\}}$$

$$\frac{}{\{l_1{:}\tau_1, \ldots, l_{i-1}{:}\tau_{i-1}, l_i{:}\tau_i, \ldots, l_n{:}\tau_n\} \leq \{l_1{:}\tau_1, \ldots, l_i{:}\tau_i, l_{i-1}{:}\tau_{i-1}, \ldots, l_n{:}\tau_n\}}$$

$$\frac{\tau_i \leq \tau_i'}{\{l_1{:}\tau_1, \ldots, l_i{:}\tau_i, \ldots, l_n{:}\tau_n\} \leq \{l_1{:}\tau_1, \ldots, l_i{:}\tau_i', \ldots, l_n{:}\tau_n\}}$$

$$\frac{\tau_3 \leq \tau_1 \qquad \tau_2 \leq \tau_4}{\tau_1 \to \tau_2 \leq \tau_3 \to \tau_4} \qquad \frac{}{\tau \leq \tau} \qquad \frac{\tau_1 \leq \tau_2 \qquad \tau_2 \leq \tau_3}{\tau_1 \leq \tau_3}$$

$$\begin{array}{rcl} e & ::= & c \mid x \mid \lambda x{:}\tau.\ e \mid e\ e \mid \Lambda\alpha.\ e \mid e[\tau] \\ \tau & ::= & \mathsf{int} \mid \tau \to \tau \mid \alpha \mid \forall\alpha.\tau \\ v & ::= & c \mid \lambda x{:}\tau.\ e \mid \Lambda\alpha.\ e \end{array} \qquad \begin{array}{rcl} \Gamma & ::= & \cdot \mid \Gamma, x{:}\tau \\ \Delta & ::= & \cdot \mid \Delta, \alpha \end{array}$$

$\boxed{e \to e' \text{ and } \Delta; \Gamma \vdash e : \tau}$

$$\frac{e \to e'}{e\ e_2 \to e'\ e_2} \qquad \frac{e \to e'}{v\ e \to v\ e'} \qquad \frac{e \to e'}{e[\tau] \to e'[\tau]} \qquad \frac{}{(\lambda x{:}\tau.\ e)v \to e[v/x]} \qquad \frac{}{(\Lambda\alpha.\ e)[\tau] \to e[\tau/\alpha]}$$

$$\frac{}{\Delta; \Gamma \vdash x : \Gamma(x)} \qquad \frac{}{\Delta; \Gamma \vdash c : \mathsf{int}} \qquad \frac{\Delta; \Gamma, x{:}\tau_1 \vdash e : \tau_2 \qquad \Delta \vdash \tau_1}{\Delta; \Gamma \vdash \lambda x{:}\tau_1.\ e : \tau_1 \to \tau_2} \qquad \frac{\Delta, \alpha; \Gamma \vdash e : \tau_1}{\Delta; \Gamma \vdash \Lambda\alpha.\ e : \forall\alpha.\tau_1}$$

$$\frac{\Delta; \Gamma \vdash e_1 : \tau_2 \to \tau_1 \qquad \Delta; \Gamma \vdash e_2 : \tau_2}{\Delta; \Gamma \vdash e_1\ e_2 : \tau_1} \qquad \frac{\Delta; \Gamma \vdash e : \forall\alpha.\tau_1 \qquad \Delta \vdash \tau_2}{\Delta; \Gamma \vdash e[\tau_2] : \tau_1[\tau_2/\alpha]}$$

Name:____

$$\begin{array}{rcl} e & ::= & \ldots \mid \mathsf{A}(e) \mid \mathsf{B}(e) \mid (\mathsf{match}\ e\ \mathsf{with}\ \mathsf{A}x.\ e \mid \mathsf{B}x.\ e) \mid \mathsf{roll}_\tau\ e \mid \mathsf{unroll}\ e \\ \tau & ::= & \ldots \mid \tau_1 + \tau_2 \mid \mu\alpha.\tau \\ v & ::= & \ldots \mid \mathsf{A}(v) \mid \mathsf{B}(v) \mid \mathsf{roll}_\tau\ v \end{array}$$

$$\frac{}{\mathsf{match}\ \mathsf{A}(v)\ \mathsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2 \to e_1[v/x]} \qquad \frac{}{\mathsf{match}\ \mathsf{B}(v)\ \mathsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2 \to e_2[v/y]}$$

$$\frac{e \to e'}{\mathsf{A}(e) \to \mathsf{A}(e')} \qquad \frac{e \to e'}{\mathsf{B}(e) \to \mathsf{B}(e')} \qquad \frac{e \to e'}{\mathsf{match}\ e\ \mathsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2 \to \mathsf{match}\ e'\ \mathsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2}$$

$$\frac{}{\mathsf{unroll}\ (\mathsf{roll}_{\mu\alpha.\tau}\ v) \to v} \qquad \frac{e \to e'}{\mathsf{roll}_{\mu\alpha.\tau}\ e \to \mathsf{roll}_{\mu\alpha\tau}\ e'} \qquad \frac{e \to e'}{\mathsf{unroll}\ e \to \mathsf{unroll}\ e'}$$

$$\frac{\Delta;\Gamma \vdash e : \tau_1 + \tau_2 \qquad \Delta;\Gamma, x{:}\tau_1 \vdash e_1 : \tau \qquad \Delta;\Gamma, y{:}\tau_2 \vdash e_2 : \tau}{\Delta;\Gamma \vdash \mathsf{match}\ e\ \mathsf{with}\ \mathsf{A}x.\ e_1 \mid \mathsf{B}y.\ e_2 : \tau}$$

$$\frac{\Delta;\Gamma \vdash e : \tau_1}{\Delta;\Gamma \vdash \mathsf{A}(e) : \tau_1 + \tau_2} \qquad \frac{\Delta;\Gamma \vdash e : \tau_2}{\Delta;\Gamma \vdash \mathsf{B}(e) : \tau_1 + \tau_2} \qquad \frac{\Delta;\Gamma \vdash e : \tau[(\mu\alpha.\tau)/\alpha]}{\Delta;\Gamma \vdash \mathsf{roll}_{\mu\alpha.\tau}\ e : \mu\alpha.\tau} \qquad \frac{\Delta;\Gamma \vdash e : \mu\alpha.\tau}{\Delta;\Gamma \vdash \mathsf{unroll}\ e : \tau[(\mu\alpha.\tau)/\alpha]}$$

Module Thread:

```
type t
val create : ('a -> 'b) -> 'a -> t
val join : t -> unit
```

Module Mutex:

```
type t
val create : unit -> t
val lock : t -> unit
val unlock : t -> unit
```

Module Event:

```
type 'a channel
type 'a event
val new_channel : unit -> 'a channel
val send : 'a channel -> 'a -> unit event
val receive : 'a channel -> 'a event
val choose : 'a event list -> 'a event
val wrap : 'a event -> ('a -> 'b) -> 'b event
val sync : 'a event -> 'a
```

Name:____

1. (**25** points) This problem uses System F extended with addition.

   (a) Give the appropriate System F typing rule for addition expressions of the form $e_1 + e_2$. (This should be easy and unrelated to the other problems.)

   (b) Consider a typing context where:

   - There are no type variables in scope.
   - x is the only term variable in scope and it has type $\forall\alpha.\alpha \rightarrow \alpha$.

   i. What does τ need to be for the program fragment $x\ [\tau]\ (\lambda y : \mathsf{int}.\ y + 7)\ 11$ to typecheck? (Recall application — of types or terms — associates to the left.)

   ii. Given your choice for τ, what type does $x\ [\tau]\ (\lambda y : \mathsf{int}.\ y + 7)\ 11$ have?

   iii. Give a typing derivation for just $x\ [\tau]$ (*not* the entire program fragment; that's too much work) using the typing context described above.

   (c) If v is an arbitrary value of type $\forall\alpha.\alpha \rightarrow \alpha$, then what might $v\ [\tau]\ (\lambda y : \mathsf{int}.\ y + 7)\ 11$ evaluate to?

   (d) If v is an arbitrary value such that $v\ (\lambda y : \mathsf{int}.\ y + 7)\ 11$ type-checks (notice v is no longer polymorphic), then:

   i. What type does v have?

   ii. What might $v\ (\lambda y : \mathsf{int}.\ y + 7)\ 11$ evaluate to?

Name:____

2. (**20** points)

   Consider a typed λ-calculus with sum types, pair types, recursive types, unit, and int.

   (a) Define a type `t1` for a binary tree of integers where:
   - Each interior node has one integer and two children.
   - Each leaf node has no data.

   (b) Give a type `t2` for a binary tree of integers where:
   - Each node has one integer and two *optional* children (meaning each child may or may not be another binary tree).

   (c) Explain in English how there is exactly one value of type `t1` that cannot be translated to an equivalent value of type `t2`.

   (d) Define the value you described in the previous problem using an actual λ-calculus expression. Make sure your value has type `t1`.

Name:____

3. (**25** points)

   Use Concurrent ML to complete an implementation of "infinite" arrays (in the sense that no index is out of bounds), without using Caml's references or arrays. More specifically, implement the `new_array` function for this code:

```
(* Interface *)
type 'a myarray
val new_array : 'a -> 'a myarray (* Initially, every index maps to 'a *)
val set : 'a myarray -> int -> 'a -> unit (* change value in index *)
val get : 'a myarray -> int -> 'a  (* return current value in index *)

(* Implementation *)
open Event
open Thread
type 'a myarray = ((int * ('a channel)) channel) * ((int * 'a) channel)

let new_array init = (* for you *)
let set (_,c) i v =
   sync (send c (i,v))
let get (c,_) i =
   let ret = new_channel() in
   sync (send c (i,ret));
   sync (receive ret)
```

   Hints: Do *not* worry about being efficient. Have `set` work in $O(1)$ time and `get` work in (worst-case) $O(n)$ time where n is the number of `set` operations preceding it. Use an association list. Sample solution is about 15 lines, including a short helper function for traversing a list. Use `choose` and `wrap`.

Name:____

4. (**20** points)

Consider a class-based OOP language like we did in class, but suppose methods do *not* have implicit access to `self` (also known as `this`). More specifically:

- As usual, subclasses inherit the fields and methods of superclasses and can override methods. The method-lookup rules are the same.
- As usual, we "confuse" classes and types, so a subclass is a subtype.
- Unlike in OOP, a method in class `C` has to take an explicit argument of type `C` to access any fields/methods in its body. For example, instead of having a method like
  `int get_sum() { return self.x + self.y }`
  and calling it like
  `e.get_sum()` (assuming the method is defined in a class `C` with fields `x` and `y`), we instead would have to do something like
  `int get_sum(C obj) { return obj.x + obj.y }`
  and call it like
  `e.get_sum(e)`.

So, this is less convenient than OOP (notice how `e.get_sum(e)` has to repeat `e`), but more flexible (since callers are not required to use the same `e` in both places).

(a) In this strange language, give an example showing how you want covariant subtyping on method arguments in order for overriding methods to use the fact that they are defined in the subclass.

(b) Show how covariant subtyping on method arguments is unsound by giving a use of your example from part (a) in which the program gets stuck even though it typechecks with covariant subtyping.

(c) In 1–3 sentences, explain why normal OOP does not have this "covariant subtyping is unsound, but contravariant subtyping is not what you want" problem.

Name:____

5. (**6** points)

   Consider a prototype-based (classless) approach to OOP as we did in class. Particularly recall how method-lookup is defined in this approach. Explain how to create an object *obj* such that any object derived from *obj* (either directly by have a parent slot holding *obj* or transitively) can have *any* method called on it without a "method not found" error occurring.

   Hints:

   - This is a very short problem.
   - Nontermination.