Name:____

CSE 505, Fall 2006, Midterm Examination
2 November 2006

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 11:50.**
- You can rip apart the pages, but please write your name on each page.
- There are **100 points** total, distributed **unevenly** among **4** questions (which have multiple parts).

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:

$$\begin{array}{rcl} s & ::= & \mathsf{skip} \mid x := e \mid s;s \mid \mathsf{if}\ e\ s\ s \mid \mathsf{while}\ e\ s \\ e & ::= & c \mid x \mid e+e \mid e*e \\ (c & \in & \{\ldots,-2,-1,0,1,2,\ldots\}) \\ (x & \in & \{\mathtt{x}_1,\mathtt{x}_2,\ldots,\mathtt{y}_1,\mathtt{y}_2,\ldots,\mathtt{z}_1,\mathtt{z}_2,\ldots,\ldots\}) \end{array}$$

$\boxed{H\ ;\ e\ \Downarrow\ c}$

$$\textsc{const}\ \frac{}{H\ ;\ c\ \Downarrow\ c} \qquad \textsc{var}\ \frac{}{H\ ;\ x\ \Downarrow\ H(x)} \qquad \textsc{add}\ \frac{H\ ;\ e_1\ \Downarrow\ c_1 \qquad H\ ;\ e_2\ \Downarrow\ c_2}{H\ ;\ e_1+e_2\ \Downarrow\ c_1+c_2} \qquad \textsc{mult}\ \frac{H\ ;\ e_1\ \Downarrow\ c_1 \qquad H\ ;\ e_2\ \Downarrow\ c_2}{H\ ;\ e_1*e_2\ \Downarrow\ c_1*c_2}$$

$\boxed{H_1\ ;\ s_1\ \rightarrow\ H_2\ ;\ s_2}$

$$\textsc{assign}\ \frac{H\ ;\ e\ \Downarrow\ c}{H\ ;\ x := e\ \rightarrow\ H, x\mapsto c\ ;\ \mathsf{skip}} \qquad \textsc{seq1}\ \frac{}{H\ ;\ \mathsf{skip}; s\ \rightarrow\ H\ ;\ s} \qquad \textsc{seq2}\ \frac{H\ ;\ s_1\ \rightarrow\ H'\ ;\ s_1'}{H\ ;\ s_1;s_2\ \rightarrow\ H'\ ;\ s_1';s_2}$$

$$\textsc{if1}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad c>0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ \rightarrow\ H\ ;\ s_1} \qquad \textsc{if2}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad c\leq 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ \rightarrow\ H\ ;\ s_2} \qquad \textsc{while}\ \frac{}{H\ ;\ \mathsf{while}\ e\ s\ \rightarrow\ H\ ;\ \mathsf{if}\ e\ (s; \mathsf{while}\ e\ s)\ \mathsf{skip}}$$

$$\begin{array}{rcl} e & ::= & \lambda x.\ e \mid x \mid e\ e \mid c \\ v & ::= & \lambda x.\ e \mid c \\ \tau & ::= & \mathsf{int} \mid \tau \rightarrow \tau \end{array}$$

$\boxed{e \rightarrow e'}$

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

$\boxed{e[e'/x] = e''}$

$$\frac{}{x[e/x] = e} \qquad \frac{e_1[e/x] = e_1' \qquad y \neq x \qquad y \notin FV(e)}{(\lambda y.\ e_1)[e/x] = \lambda y.\ e_1'}$$

$$\frac{y \neq x}{y[e/x] = y} \qquad \frac{e_1[e/x] = e_1' \qquad e_2[e/x] = e_2'}{(e_1\ e_2)[e/x] = e_1'\ e_2'}$$

$\boxed{\Gamma \vdash e : \tau}$

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

- If $\cdot \vdash e : \tau$ and $e \rightarrow e'$, then $\cdot \vdash e' : \tau$.
- If $\cdot \vdash e : \tau$, then e is a value or there exists an e' such that $e \rightarrow e'$.
- If $\Gamma, x{:}\tau' \vdash e : \tau$ and $\Gamma \vdash e' : \tau'$, then $\Gamma \vdash e[e'/x] : \tau$.

Name:____

1. (IMP with booleans)
In this problem we extend the IMP expression language with booleans: true, false, negation, and inclusive-or. (Variables hold integers or booleans, but that is not directly relevant to the questions below.) The new syntax forms are:

$$e \quad ::= \quad \ldots \mid \mathsf{true} \mid \mathsf{false} \mid \neg e \mid e \vee e$$

The result of evaluating an expression can be an integer (not relevant below), true, or false. That is, we have $H \; ; \; e \; \Downarrow \; v$ where $v ::= c \mid \mathsf{true} \mid \mathsf{false}$.

Negation and inclusive-or can be "stuck" if a subexpression does not evaluate to a boolean.

(a) (**10** points) Add rules to our large-step operational semantics to support the new syntax forms. For $e_1 \vee e_2$, use short-circuiting left-to-right evaluation (like `||` in many languages). If your rules all contain explicit uses of false and true, then you should expect to write 7 rules.

(b) (**12** points) Theorem: If e always evaluates to a boolean, then e and $\neg\neg e$ are equivalent.

- Restate this theorem formally.
- Prove this theorem formally.

(c) (**10** points) Add implication ($e \Rightarrow e$) to the language. Recall "a implies b if a is false or b is true."

- Give large-step operational semantics rules that support this extension "directly," using short-circuiting left-to-right evaluation. If your rules all contain explicit uses of false and true, then you should expect to write 3 rules.
- Give 1 rule that works just as well as your 3 rules by treating implication as a derived form. Remember this should be a large-step rule. Use v in this rule.

Solution:

(a)

$$\frac{}{H \; ; \; \mathsf{true} \; \Downarrow \; \mathsf{true}} \qquad \frac{}{H \; ; \; \mathsf{false} \; \Downarrow \; \mathsf{false}} \qquad \frac{H \; ; \; e \; \Downarrow \; \mathsf{true}}{H \; ; \; \neg e \; \Downarrow \; \mathsf{false}} \qquad \frac{H \; ; \; e \; \Downarrow \; \mathsf{false}}{H \; ; \; \neg e \; \Downarrow \; \mathsf{true}}$$

$$\frac{H \; ; \; e_1 \; \Downarrow \; \mathsf{true}}{H \; ; \; e_1 \vee e_2 \; \Downarrow \; \mathsf{true}} \qquad \frac{H \; ; \; e_1 \; \Downarrow \; \mathsf{false} \qquad H \; ; \; e_2 \; \Downarrow \; \mathsf{true}}{H \; ; \; e_1 \vee e_2 \; \Downarrow \; \mathsf{true}} \qquad \frac{H \; ; \; e_1 \; \Downarrow \; \mathsf{false} \qquad H \; ; \; e_2 \; \Downarrow \; \mathsf{false}}{H \; ; \; e_1 \vee e_2 \; \Downarrow \; \mathsf{false}}$$

(b) Assume for all H that $H \; ; \; e \; \Downarrow \; \mathsf{true}$ or $H \; ; \; e \; \Downarrow \; \mathsf{false}$. Then for all H, e, and v, $H \; ; \; e \; \Downarrow \; v$ if and only if $H \; ; \; \neg\neg e \; \Downarrow \; v$. We prove the two directions separately. First assume $H \; ; \; e \; \Downarrow \; v$. Because v is true or false, one of these derivations suffices to show $H \; ; \; \neg\neg e \; \Downarrow \; v$:

$$\frac{\dfrac{H \; ; \; e \; \Downarrow \; \mathsf{true}}{H \; ; \; \neg e \; \Downarrow \; \mathsf{false}}}{H \; ; \; \neg\neg e \; \Downarrow \; \mathsf{true}} \qquad \frac{\dfrac{H \; ; \; e \; \Downarrow \; \mathsf{false}}{H \; ; \; \neg e \; \Downarrow \; \mathsf{true}}}{H \; ; \; \neg\neg e \; \Downarrow \; \mathsf{false}}$$

Now assume $H \; ; \; \neg\neg e \; \Downarrow \; v$. By inspection of the semantic rules, v is true or false. In the former case, we can derive this only if $H \; ; \; \neg e \; \Downarrow \; \mathsf{false}$, which in turn we can derive only if $H \; ; \; e \; \Downarrow \; \mathsf{true}$, as desired. Similarly, in the latter case, we can derive this only if $H \; ; \; \neg e \; \Downarrow \; \mathsf{true}$, which in turn we can derive only if $H \; ; \; e \; \Downarrow \; \mathsf{false}$, as desired.

(c)

$$\frac{H\ ;\ e_1\ \Downarrow\ \mathsf{false}}{H\ ;\ e_1 \Rightarrow e_2\ \Downarrow\ \mathsf{true}} \qquad \frac{H\ ;\ e_1\ \Downarrow\ \mathsf{true} \qquad H\ ;\ e_2\ \Downarrow\ \mathsf{true}}{H\ ;\ e_1 \Rightarrow e_2\ \Downarrow\ \mathsf{true}} \qquad \frac{H\ ;\ e_1\ \Downarrow\ \mathsf{true} \qquad H\ ;\ e_2\ \Downarrow\ \mathsf{false}}{H\ ;\ e_1 \Rightarrow e_2\ \Downarrow\ \mathsf{false}}$$

$$\frac{H\ ;\ (\neg e_1) \vee e2\ \Downarrow\ v}{H\ ;\ e_1 \Rightarrow e_2\ \Downarrow\ v}$$

Name:____

(This page intentionally blank.)

Name:____

2. (**18** points) (IMP with large-step semantics)

   We can give IMP statements a large-step semantics with a judgment of the form $H; s \Downarrow H'$. The rules below do so, but there are *errors*. (The rules match neither our informal understanding nor our small-step semantics.) Find **three** errors (two of which are the same conceptual error), explain the problem, why it is a problem, and how to change the rules to solve the problem.

$$\textsc{skip}\quad \frac{}{H; \mathsf{skip} \Downarrow H} \qquad \textsc{assign}\quad \frac{H\ ;\ e\ \Downarrow\ c}{H; x := e \Downarrow H, x \mapsto c} \qquad \textsc{seq}\quad \frac{H; s_1 \Downarrow H_1 \qquad H; s_2 \Downarrow H_2}{H; (s_1; s_2) \Downarrow H_2}$$

$$\textsc{if1}\quad \frac{H\ ;\ e\ \Downarrow\ c \qquad H; s_1 \Downarrow H_1 \qquad H; s_2 \Downarrow H_2 \qquad c > 0}{H; \mathsf{if}\ e\ s_1\ s_2 \Downarrow H_1}$$

$$\textsc{if2}\quad \frac{H\ ;\ e\ \Downarrow\ c \qquad H; s_1 \Downarrow H_1 \qquad H; s_2 \Downarrow H_2 \qquad c \le 0}{H; \mathsf{if}\ e\ s_1\ s_2 \Downarrow H_2} \qquad \textsc{while}\quad \frac{H; \mathsf{if}\ e\ (s; \mathsf{while}\ e\ s)\ \mathsf{skip} \Downarrow H'}{H; \mathsf{while}\ e\ s \Downarrow H'}$$

Solution:

In the sequence rule, we evaulate the s_2 using the original heap. This "forgets" any assignments that s_1 may have done. The fix is to evaluate s_2 using H_1, as in this rule:

$$\textsc{seq}\quad \frac{H; s_1 \Downarrow H_1 \qquad H_1; s_2 \Downarrow H_2}{H; (s_1; s_2) \Downarrow H_2}$$

In the if rules, we evaluate both statements. This is incorrect if the "branch not taken" has an infinite loop (or could otherwise get stuck, but in IMP there are no stuck states). For example, we cannot derive $H; \mathsf{if}\ 1\ \mathsf{skip}\ (\mathsf{while}\ 1\ \mathsf{skip}) \Downarrow H$ but we would like to. This is how to fix the rules:

$$\textsc{if1}\quad \frac{H\ ;\ e\ \Downarrow\ c \qquad H; s_1 \Downarrow H_1 \qquad c > 0}{H; \mathsf{if}\ e\ s_1\ s_2 \Downarrow H_1} \qquad \textsc{if2}\quad \frac{H\ ;\ e\ \Downarrow\ c \qquad H; s_2 \Downarrow H_2 \qquad c \le 0}{H; \mathsf{if}\ e\ s_1\ s_2 \Downarrow H_2}$$

Name:____

3. (**15** points) (Caml and functional programming)

   Consider this Caml code, which type-checks and runs correctly.

```
type dumbTree = Empty | Node of dumbTree * dumbTree

let rec s f t =
   match t with
     Empty -> f t
   | Node(x,y) -> f t + s f x + s f y

let c1 t = s (fun x -> 1) t
let c2 t = s (fun x -> match x with Node(l,Empty) -> 1 | _ -> 0) t
```

   (a) What are the types of `s`, `c1`, and `c2`?

   (b) What do `c1` and `c2` compute? (Hint: The answers are straightforward.)

   (c) Rewrite the last two lines of the code so they are shorter and equivalent.

   Solution:

   (a)
```
val s : (dumbTree -> int) -> dumbTree -> int
val c1 : dumbTree -> int
val c2 : dumbTree -> int
```

   (b) `c1` takes a tree and returns the total number of internal nodes plus leaves it has. `c2` takes a tree and returns how many of its internal nodes have right-children that are leaves (i.e., `Empty`).

   (c)
```
let c1 = s (fun x -> 1)
let c2 = s (fun x -> match x with Node(l,Empty) -> 1 | _ -> 0)
```

Name:____

4. (Coin-flipping in Lambda-Calculus)

In this problem we take the simply-typed lambda-calculus with conditionals (true, false, if e_1 e_2 e_3, and the type bool) and add a "coin-flip" expression, flip. This expression is not a value. Our call-by-value left-to-right small-step semantics has two new semantic rules:

$$\frac{}{\mathsf{flip} \rightarrow \mathsf{true}} \qquad\qquad \frac{}{\mathsf{flip} \rightarrow \mathsf{false}}$$

(a) (**5** points) In lambda-calculus with conditionals, write a (curried) function that returns the exclusive-or of its arguments. Do not use the constant true and use the constant false only once. (This does not require flip.)

(b) (**5** points) Argue that for all e, $(\lambda x.\ e)$ true and e[true$/x$] are equivalent under call-by-value.

(c) (**8** points) Argue that depending on e, $(\lambda x.\ e)$ flip and e[flip$/x$] may or may not be equivalent under call-by-value.

(d) (**5** points) Give a typing rule for flip.

(e) (**12** points) Assuming we have proofs of progress, preservation, and substitution for lambda-calculus with conditionals, explain how to extend the proofs for programs containing flip. Be clear about the induction hypothesis and what cases you are adding.

Solution:

(a) $\lambda x.\ \lambda y.$ if x (if y false x) y

(b) Given $(\lambda x.\ e)$ true, only one evaluation rule applies and produces e[true$/x$]. So the expressions can produce exactly the same results (or non-termination); $(\lambda x.\ e)$ true just takes one more step.

(c) An example where they are equivalent is $e = x$; both expressions always terminate and evaluate to true or false. An example where they are not equivalent is when $\lambda x.\ e$ is our "xor" function from part (a). If we apply "xor" to flip under call-by-value we get either a function equivalent to negation (if flip $\rightarrow$ true) or the identity function (if flip $\rightarrow$ false). But the body of "xor" with flip substituted for x is a function that when passed false could return true or false. (There are simpler examples; the point is e needs to use x more than once.)

(d)

$$\frac{}{\Gamma \vdash \mathsf{flip} : \mathsf{bool}}$$

(e)
- Preservation: We show if $\cdot \vdash e : \tau$ and $e \rightarrow e'$, then $\cdot \vdash e' : \tau$ by induction on the derivation of $\cdot \vdash e : \tau$. The one new case is when e is flip, so τ is bool. By inspecting the operational semantics, two rules apply and e' is true or false. In either case, we have axioms that let us derive $\cdot \vdash$ true : bool and $\cdot \vdash$ false : bool.
- Progress: We show if $\cdot \vdash e : \tau$, then e is a value or there exists an e' such that $e \rightarrow e'$ by induction on the derivation of $\cdot \vdash e : \tau$. The one new case is when e is flip. In this case, we can always take a step, for example let e' be true.
- Substitution: We show if $\Gamma, x{:}\tau' \vdash e : \tau$ and $\Gamma \vdash e' : \tau'$, then $\Gamma \vdash e[e'/x] : \tau$ by induction on the derivation of $\Gamma, x{:}\tau' \vdash e : \tau$. The one new case is when e is flip so τ is bool. In this case, $e[e'/x]$ is also flip and we can derive $\Gamma \vdash$ flip : bool.

Name:______________________________________

(This page intentionally blank.)