Name:____

CSE 505, Fall 2009, Midterm Examination
5 November 2009

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 **5** 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:

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

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

$$\text{CONST}\ \frac{}{H\ ;\ c\ \Downarrow\ c} \qquad \text{VAR}\ \frac{}{H\ ;\ x\ \Downarrow\ H(x)} \qquad \text{ADD}\ \frac{H\ ;\ e_1\ \Downarrow\ c_1 \qquad H\ ;\ e_2\ \Downarrow\ c_2}{H\ ;\ e_1 + e_2\ \Downarrow\ c_1 + c_2} \qquad \text{MULT}\ \frac{H\ ;\ e_1\ \Downarrow\ c_1 \qquad H\ ;\ e_2\ \Downarrow\ c_2}{H\ ;\ e_1 * e_2\ \Downarrow\ c_1 * c_2}$$

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

$$\text{ASSIGN}\ \frac{H\ ;\ e\ \Downarrow\ c}{H\ ;\ x := e\ \rightarrow\ H, x \mapsto c\ ;\ \mathsf{skip}} \qquad \text{SEQ1}\ \frac{}{H\ ;\ \mathsf{skip}; s\ \rightarrow\ H\ ;\ s} \qquad \text{SEQ2}\ \frac{H\ ;\ s_1\ \rightarrow\ H'\ ;\ s_1'}{H\ ;\ s_1; s_2\ \rightarrow\ H'\ ;\ s_1'; s_2}$$

$$\text{IF1}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad c > 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ \rightarrow\ H\ ;\ s_1} \qquad \text{IF2}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad c \leq 0}{H\ ;\ \mathsf{if}\ e\ s_1\ s_2\ \rightarrow\ H\ ;\ s_2} \qquad \text{WHILE}\ \frac{}{H\ ;\ \mathsf{while}\ e\ s\ \rightarrow\ H\ ;\ \mathsf{if}\ e\ (s; \mathsf{while}\ e\ s)\ \mathsf{skip}}$$

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

$\boxed{e \rightarrow e'}$

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

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

$$\frac{}{x[e/x] = e} \qquad \frac{y \neq x}{y[e/x] = y} \qquad \frac{}{c[e/x] = c}$$

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

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

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

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

Name:____

1. In this problem, we change IMP by adding one more *constant*, $\square$:

$$c \quad \in \quad \{\ldots, -2, -1, 0, 1, 2, \ldots\} \cup \{\square\}$$

Because $\square$ is a constant, it can also be an expression, the result of evaluating an expression, or the contents of a heap variable. However, $\square$ is *not* a legal argument to any "math" operators like "(blue) plus" except "$=$" and "$\neq$".

Informally, if an expression has any subexpression that evaluates to $\square$, then the expression evaluates to $\square$.

(a) (**7** points) Add four inference rules to the $H \; ; \; e \; \Downarrow \; c$ judgment to account for $\square$.

(b) (**17** points) Considering all the inference rules now in the language, prove that if e contains a $\square$ and $H \; ; \; e \; \Downarrow \; c$, then c is $\square$. Hint: Use induction. The new rules from part (a) are *not* the difficult cases.

(c) (**6** points) Our IMP *statement* semantics can now get stuck. In English, explain exactly how this could occur. Propose a small change to the statement semantics to avoid this. Give any new inference rule(s) and explain in English how you changed the meaning of the language.

Solution:

(a)

$$\textsc{sq1}\ \frac{H \; ; \; e_1 \; \Downarrow \; \square}{H \; ; \; e_1 + e_2 \; \Downarrow \; \square} \qquad \textsc{sq2}\ \frac{H \; ; \; e_2 \; \Downarrow \; \square}{H \; ; \; e_1 + e_2 \; \Downarrow \; \square} \qquad \textsc{sq3}\ \frac{H \; ; \; e_1 \; \Downarrow \; \square}{H \; ; \; e_1 * e_2 \; \Downarrow \; \square} \qquad \textsc{sq4}\ \frac{H \; ; \; e_2 \; \Downarrow \; \square}{H \; ; \; e_1 * e_2 \; \Downarrow \; \square}$$

(b) Proof by induction on the derivation of $H \; ; \; e \; \Downarrow \; c$ proceeding by the rules instantiated at the bottom of the derivation:

CONST Then $e = c$, so if e contains $\square$, then e and c are $\square$.

VAR Holds vacuously because e is some x and therefore does not contain $\square$.

ADD Because the result is $c_1 + c_2$, neither c_1 nor c_2 is $\square$. Therefore, since $H \; ; \; e_1 \; \Downarrow \; c_1$ and $H \; ; \; e_2 \; \Downarrow \; c_2$, by induction neither e_1 nor e_2 contains $\square$. So $e_1 + e_2$ does not contain $\square$ and the theorem holds vacuously.

MULT Because the result is $c_1 * c_2$, neither c_1 nor c_2 is $\square$. Therefore, since $H \; ; \; e_1 \; \Downarrow \; c_1$ and $H \; ; \; e_2 \; \Downarrow \; c_2$, by induction neither e_1 nor e_2 contains $\square$. So $e_1 * e_2$ does not contain $\square$ and the theorem holds vacuously.

SQ1 Holds trivially because c is $\square$.

SQ2 Holds trivially because c is $\square$.

SQ3 Holds trivially because c is $\square$.

SQ4 Holds trivially because c is $\square$.

Note: The proof can also be done by structural induction on e, but then you need to argue for each form of e what rules could apply.

(c) A statement can get stuck if the expression in an if-statement evaluates to $\square$ because it is neither the case that $\square > 0$ nor $\square \leq 0$. There are of course many ways to fix this. We could add the rule below, which has the effect of treating $\square$ as false, just like a non-positive number.

$$\textsc{if3}\ \frac{H \; ; \; e \; \Downarrow \; \square}{H \; ; \; \mathsf{if}\ e\ s_1\ s_2 \; \rightarrow \; H \; ; \; s_2}$$

Name:____

(This page intentionally blank)

Name:____

2. In this problem, we use the following ***large-step*** semantics for IMP statements: The judgment is $H\ ;\ s\ \Downarrow\ H'$ meaning s under heap H produces heap H'. The inference rules are:

$$\text{SKIP}\ \frac{}{H;\mathsf{skip}\Downarrow H} \qquad \text{ASSIGN}\ \frac{H\ ;\ e\ \Downarrow\ c}{H;x:=e\Downarrow H,x\mapsto c} \qquad \text{SEQ}\ \frac{H;s_1\Downarrow H_1 \qquad H_1;s_2\Downarrow H_2}{H;(s_1;s_2)\Downarrow H_2}$$

$$\text{IF1}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad H;s_1\Downarrow H_1 \qquad c>0}{H;\mathsf{if}\ e\ s_1\ s_2\Downarrow H_1} \qquad \text{IF2}\ \frac{H\ ;\ e\ \Downarrow\ c \qquad H;s_2\Downarrow H_2 \qquad c\le 0}{H;\mathsf{if}\ e\ s_1\ s_2\Downarrow H_2}$$

$$\text{WHILE}\ \frac{H;\mathsf{if}\ e\ (s;\mathsf{while}\ e\ s)\ \mathsf{skip}\Downarrow H'}{H;\mathsf{while}\ e\ s\Downarrow H'}$$

The sequence operator is associative. That is, $s_1;(s_2;s_3)$ and $(s_1;s_2);s_3$ are equivalent.

(a) (**5** points) State this associativity fact formally as a theorem in terms of the large-step semantics for statements.

(b) (**16** points) Prove the theorem you stated in part (a). Hint: Do *not* use induction.

Solution:

(a) (For all H, H', s_1, s_2, and s_3), $H;(s_1;(s_2;s_3))\Downarrow H'$ if and only if $H;((s_1;s_2);s_3)\Downarrow H'$.

(b) We prove the two directions of the if and only if separately.

First assume $H;(s_1;(s_2;s_3))\Downarrow H'$. Inverting the derivation ensures we have a derivation that looks like this for some H_1 and H_2:

$$\dfrac{\dfrac{\vdots}{H;s_1\Downarrow H_1} \qquad \dfrac{\dfrac{\vdots}{H_1;s_2\Downarrow H_2} \qquad \dfrac{\vdots}{H_2;s_3\Downarrow H'}}{H_1;(s_2;s_3)\Downarrow H'}}{H;(s_1;(s_2;s_3))\Downarrow H'}$$

So we know $H;s_1\Downarrow H_1$, $H_1;s_2\Downarrow H_2$, and $H_2;s_3\Downarrow H'$, from which we can derive:

$$\dfrac{\dfrac{H;s_1\Downarrow H_1 \qquad H_1;s_2\Downarrow H_2}{H;(s_1;s_2)\Downarrow H_2} \qquad H_2;s_3\Downarrow H'}{H;((s_1;s_2);s_3)\Downarrow H'}$$

Now assume $H;((s_1;s_2);s_3)\Downarrow H'$. Inverting the derivation ensures we have a derivation that looks like this for some H_1 and H_2:

$$\dfrac{\dfrac{\dfrac{\vdots}{H;s_1\Downarrow H_1} \qquad \dfrac{\vdots}{H_1;s_2\Downarrow H_2}}{H;(s_1;s_2)\Downarrow H_2} \qquad \dfrac{\vdots}{H_2;s_3\Downarrow H'}}{H;((s_1;s_2);s_3)\Downarrow H'}$$

So we know $H;s_1\Downarrow H_1$, $H_1;s_2\Downarrow H_2$, and $H_2;s_3\Downarrow H'$, from which we can derive:

$$\dfrac{H;s_1\Downarrow H_1 \qquad \dfrac{H_1;s_2\Downarrow H_2 \qquad H_2;s_3\Downarrow H'}{H_1;(s_2;s_3)\Downarrow H'}}{H;(s_1;(s_2;s_3))\Downarrow H'}$$

Name:____

(This page intentionally blank)

Name:____

3. (**14** points) Describe what, if anything, each of the following Caml programs would print:

(a)
```
let f x y = x y in
let z = f print_string "hi " in
f print_string "hi"
```

(b)
```
let f x = (fun y -> print_string x) in
let g = f "elves " in
let x = "trees " in
g "cookies "
```

(c)
```
let rec f n x =
  if n>=0
  then (let _ = print_string x in f (n-1) x)
  else ()
in
f 3 "hi "
```

(d)
```
let rec f n x =
  if n>=0
  then (let _ = print_string x in f (n-1) x)
  else ()
in
f 3
```

(e)
```
let rec f x = f x in
print_string (f "hi ")
```

Solution:

(a) hi hi

(b) elves

(c) hi hi hi hi

(d) prints nothing (evaluates to a function that prints when called)

(e) prints nothing (goes into an infinite loop)

Name:____

4. In this problem, we use the untyped lambda calculus with small-step call-by-value left-to-right evaluation. Recall this encoding of pairs:

   "mkpair" $\lambda x.\ \lambda y.\ \lambda z.\ z\ x\ y$

   "fst" $\lambda p.\ p(\lambda x.\ \lambda y.\ x)$

   "snd" $\lambda p.\ p(\lambda x.\ \lambda y.\ y)$

   (a) (**9** points) For any values v_1 and v_2, "fst" ("mkpair" v_1 v_2) produces a value in 6 steps. Writing only lambda terms (i.e., no abbreviations), show these steps. Show just the result of each step, not the derivation that produces it.

   $(\lambda p.\ p\ (\lambda x.\ \lambda y.\ x))\ ((\lambda x.\ \lambda y.\ \lambda z.\ z\ x\ y)\ v_1\ v_2)$

   $\to$ ____

   $\to$ ____

   $\to$ ____

   $\to$ ____

   $\to$ ____

   $\to$ ____

   (b) (**6** points) Again using no abbreviations, extend the encoding to include a "swap" function. Given an encoding of the pair (v_1, v_2), "swap" should return an encoding of the pair (v_2, v_1).

Solution:

(a) $(\lambda p.\ p\ (\lambda x.\ \lambda y.\ x))\ ((\lambda x.\ \lambda y.\ \lambda z.\ z\ x\ y)\ v_1\ v_2)$
$\to (\lambda p.\ p\ (\lambda x.\ \lambda y.\ x))\ ((\lambda y.\ \lambda z.\ z\ v_1\ y)\ v_2)$
$\to (\lambda p.\ p\ (\lambda x.\ \lambda y.\ x))\ (\lambda z.\ z\ v_1\ v_2)$
$\to (\lambda z.\ z\ v_1\ v_2)\ (\lambda x.\ \lambda y.\ x)$
$\to (\lambda x.\ \lambda y.\ x)\ v_1\ v_2$
$\to (\lambda y.\ v_1)\ v_2$
$\to v_1$

(b) There are an infinite number of correct solutions. Here are four:

- $\lambda p.\ (\lambda x.\ \lambda y.\ \lambda z.\ z\ x\ y)(p\ \lambda x.\ \lambda y.\ y)(p\ \lambda x.\ \lambda y.\ x)$
- $\lambda p.\ \lambda z.\ z\ (p\ \lambda x.\ \lambda y.\ y)(p\ \lambda x.\ \lambda y.\ x)$
- $\lambda p.\ (\lambda x.\ \lambda y.\ \lambda z.\ z\ y\ x)(p\ \lambda x.\ \lambda y.\ x)(p\ \lambda x.\ \lambda y.\ y)$
- $\lambda p.\ p\ \lambda x.\ \lambda y.\ \lambda z.\ z\ y\ x$

Name:____

5. In this problem, we consider the simply-typed lambda-calculus (using small-step call-by-value left-to-right evaluation). We suppose the integer constants c (of type int) include only positive integers (1, 2, 3, ...), i.e., we remove negative numbers. We add a subtraction operator ($e ::= \ldots \mid e - e$) and these rules:

$$\frac{c_3 \text{ is math's subtraction of } c_2 \text{ from } c_1}{c_1 - c_2 \rightarrow c_3} \qquad \frac{\Gamma \vdash e_1 : \mathsf{int} \qquad \Gamma \vdash e_2 : \mathsf{int}}{\Gamma \vdash e_1 - e_2 : \mathsf{int}}$$

   (a) (**4** points) Our operational semantics needs two additional rules. Give them.

   (b) (**6** points) Our language is *not* type-safe. Demonstrate this.

   (c) (**10** points) Consider the Preservation Lemma, the Progress Lemma, and the Substitution Lemma. Which of these lemmas are true in our language? Explain your answers briefly, but proofs are not required.

Solution:

(a)

$$\frac{e_1 \rightarrow e_1'}{e_1 - e_2 \rightarrow e_1' - e_2} \qquad \frac{e_2 \rightarrow e_2'}{v - e_2 \rightarrow v - e_2'}$$

(b) Consider an expression like 3 - 4. It type-checks under the emtpy context $(\cdot)$ with type int, but it cannot take a step because the result of the mathematical subtraction is -1, which is not in our language, so no rule applies.

(c) The Progress Lemma does *not* hold; see part (b).

The Preservation Lemma *does* hold: all three new operational rules produce an expression of type int assuming the expression before the step type-checks with type int.

The Substitution Lemma *does* hold (assuming $(e_1 - e_2)[e/x] = (e_1[e/x]) - (e_2[e/x])$); the case of the proof for subtractions expressions follows from induction just like the case for application.