Name:____

CSE 505, Fall 2008, Midterm Examination
29 October 2008

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 1:20.**
- 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:

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

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

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

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

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

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

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

$\boxed{e \rightarrow e'}$

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

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

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

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

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

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

- 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. This problem (and the next one) involve a tiny language for, "moving around the integer number line."

   Syntax:

$$
\begin{array}{rcl}
e & ::= & \mathsf{go}\ c \mid \mathsf{reverse} \mid e; e \\
d & ::= & \mathsf{R} \mid \mathsf{L} \\
(c & \in & \{\ldots, -2, -1, 0, 1, 2, \ldots\})
\end{array}
$$

   Informal semantics: A "position" is a direction d and an intger c. The direction R is "to the right," i.e., toward greater numbers, and L is "to the left," i.e., toward lesser numbers. An expression "changes the position" as follows:

   - $\mathsf{go}\ c$ has no effect on the position, if c is negative.
   - $\mathsf{go}\ c$ moves the position distance c *in the current direction* and does not change the direction, if c is non-negative. For example, if the "position" is $\mathsf{R}; 3$, then $\mathsf{go}\ 2$ produces position $\mathsf{R}; 5$.
   - $\mathsf{reverse}$ changes the direction only.
   - $e_1; e_2$ does e_1 and then e_2.

   (a) (**12** points) Give a **large-step** operational semantics for our language. The judgment should have the form $d; c; e \Downarrow d'; c'$ where d and c comprise the "starting position" and d' and c' comprise the "ending position." Do not use any other definitions, functions, or judgments. Hint: Sample solution uses 6 rules and uses "math's" $+$, $-$, $<$, and $\geq$.

   (b) (**14** points) Prove this theorem: If e has no $\mathsf{reverse}$ expressions in it and $\mathsf{R}; c; e \Downarrow d'; c'$, then $c' \geq c$. You need a stronger induction hypothesis; be sure to state it clearly.

   (c) (**4** points) Describe exactly where your proof in part (b) relies on the stronger induction hypothesis.

Solution:

(a)

$$
\textsc{neg}\ \frac{c' < 0}{d; c; \mathsf{go}\ c' \Downarrow d; c}
\qquad
\textsc{gor}\ \frac{c \geq 0}{\mathsf{R}; c; \mathsf{go}\ c' \Downarrow \mathsf{R}; c + c'}
\qquad
\textsc{gol}\ \frac{c \geq 0}{\mathsf{L}; c; \mathsf{go}\ c' \Downarrow \mathsf{L}; c - c'}
\qquad
\textsc{revr}\ \frac{}{\mathsf{R}; c; \mathsf{reverse} \Downarrow \mathsf{L}; c}
$$

$$
\textsc{revl}\ \frac{}{\mathsf{L}; c; \mathsf{reverse} \Downarrow \mathsf{R}; c}
\qquad
\textsc{seq}\ \frac{d_0; c_0; e_1 \Downarrow d_1; c_1 \qquad d_1; c_1; e_2 \Downarrow d_2; c_2}{d_0; c_0; (e_1; e_2) \Downarrow d_2; c_2}
$$

(b) Stronger induction hypothesis: If e has no $\mathsf{reverse}$ expressions in it and $\mathsf{R}; c; e \Downarrow d'; c'$, then $c' \geq c$ and $d' = \mathsf{R}$. Proof is by induction on the derivation of $\mathsf{R}; c; e \Downarrow d'; c'$, proceeding by cases on the bottommost rule used in the derivation:

- NEG Then $d' = \mathsf{R}$ anc $c' = c$ because this rule does not change the position.
- GOR Then $d' = \mathsf{R}$ (clearly in the rule) and $c' \geq c$ because c' is c plus a nonnegative number.
- GOL This rule cannot be used because the initial direction is R.
- REVR This rule cannote be used because e has no $\mathsf{reverse}$ expressions, so e cannot be $\mathsf{reverse}$.
- REVL This rule cannot be used because e has no $\mathsf{reverse}$ expressions, so e cannot be $\mathsf{reverse}$. It also cannot be used because the initial direction is R.
- SEQ By inversion of the rule, there exists e_1, e_2, d_1, and c_1 such that e is $e_1; e_2$, $\mathsf{R}; c; e_1 \Downarrow d_1; c_1$, and $d_1; c_1; e_2 \Downarrow d'; c'$. Applying induction to $\mathsf{R}; c; e_1 \Downarrow d_1; c_1$ ensures d_1 is R anc $c_1 \geq c$. Since d_1 is R, we can apply induction to $d_1; c_1; e_2 \Downarrow d'; c'$ to ensure d' is R and $c' \geq c_1$. Since $\geq$ is transitive $c' \geq c$. With that and d' is R, we are done.

(c) In the SEQ case, without the stronger induction hypothesis we do not know d_1 is R, but we need this to apply induction to $d_1; c_1; e_2 \Downarrow d'; c'$.

Name:____

(This page intentionally blank)

Name:____

2. In this problem, you will write a Caml interpreter for the language in the previous problem.

   (a) (**3** points) Give two Caml `type` definitions to define expressions e and directions d. (For constants, use the Caml `int` type.)

   (b) (**12** points) Write a function `interp` that takes two curried arguments: (1) a *pair* that is the "starting position" and (2) an expression. Return the pair that is the "ending position."

   (c) (**2** points) Give the Caml type of `interp`.

   (d) (**2** points) Implement `interp_from_origin`, which takes just an expression and returns the result of running it from an initial position that is at the origin of the number line and facing right. Use partial application.

   (e) (**1** points) Give the Caml type of `interp_from_origin`.

Solution:

(a)
```
type exp = Go of int | Reverse | Seq of exp * exp
type dir = R | L
```

(b)
```
(* Note a solution with nested match expressions is also fine *)
let rec interp (d,c) e =
   match d,e with
      R, Go c' -> if c' < 0 then (d,c) else (R,c+c')
    | L, Go c' -> if c' < 0 then (d,c) else (L,c-c')
    | R, Reverse -> (L,c)
    | L, Reverse -> (R,c)
    | _, Seq(e1,e2) -> interp (interp (d,c) e1) e2
```

(c) `(dir*int) -> exp -> (dir*int)`

(d) `let interp_from_origin = interp (R,0)`

(e) `exp -> (dir*int)`

Name:____

3. This problem extends IMP statements with this strange new syntax and small-step evaluation rules:

$$s ::= \ \ldots \mid \ s \,\#\, s$$

$$\frac{}{H \ ; \ \mathsf{skip} \,\#\, s \ \rightarrow \ H \ ; \ s} \qquad\qquad \frac{H \ ; \ s_1 \ \rightarrow \ H' \ ; \ s_1'}{H \ ; \ s_1 \,\#\, s_2 \ \rightarrow \ H' \ ; \ s_2 \,\#\, s_1'}$$

(a) (**6** points) Explain in 1–3 informal but precise English sentences the meaning of $s_1 \# s_2$.

(b) (**3** points) Is IMP still deterministic? Explain briefly.

(c) (**3** points) Give an H, s_1, and s_2 such that $H; s_1$ terminates, $H; s_2$ terminates, $H; s_1; s_2$ terminates, but $H; s_1 \# s_2$ does not terminate.

(d) (**3** points) Give an H, s_1, and s_2 such that $H; s_1$ terminates, $H; s_2$ terminates, $H; s_1 \# s_2$ terminates, but $H; s_1; s_2$ does not terminate.

Solution:

(a) $s_1 \# s_2$ executes s_1 and s_2 by alternating which substatement takes the next step, starting with s_1. In other words, it interleaves their execution with "time slices" of one execution step. After one statement reaches skip, the other statement finishes executing.

(b) Yes, in fact it is still the case that for all H and s, either s is skip or there is exactly one derivation of an execution step.

(c) One answer: Let H be $\cdot$, let s_1 be $x := 1; \mathsf{while}\ y\ \mathsf{skip}$, and let s_2 be $y := 1$.

(d) One answer: Let H be $\cdot$, let s_1 be $y := 0; y := 0; y := 1$, and let s_2 be $\mathsf{while}\ y\ \mathsf{skip}$. (Interestingly, changing s_1 to $y := 0; y := 1$ is still correct, but changing s_1 to $\mathsf{skip}; y := 1$ is incorrect, even though $H(y) = 0$.

Name:____

4. In this problem we add options (like Caml's `None` and `Some`) to the simply-typed λ-calculus, using a "get" primitive instead of pattern-matching.

   Syntax:

$$\begin{array}{rcl} e & ::= & \ldots \mid \mathsf{None} \mid \mathsf{Some}\ e \mid \mathsf{get}\ e \\ v & ::= & \ldots \mid \mathsf{None} \mid \mathsf{Some}\ v \\ \tau & ::= & \ldots \mid \tau\ \mathsf{option} \end{array}$$

   Operational Semantics:

$$\textsc{some}\ \frac{e \to e'}{\mathsf{Some}\ e \to \mathsf{Some}\ e'} \qquad \textsc{get-e}\ \frac{e \to e'}{\mathsf{get}\ e \to \mathsf{get}\ e'} \qquad \textsc{get-some}\ \frac{}{\mathsf{get}\ (\mathsf{Some}\ v) \to v} \qquad \textsc{get-none}\ \frac{}{\mathsf{get}\ \mathsf{None} \to \mathsf{get}\ \mathsf{None}}$$

   (a) (**4** points) One of the evaluation rules is strange and probably not what you would implement in an actual language. Which rule? What does the rule mean?

   (b) (**12** points) Give 3 appropriate new typing rules, one for each new form of expression. Your rules should be sound without being unnecessarily restrictive.

   (c) (**4** points) State the new case of the Canonical Forms Lemma for values of option types. (You do not need to prove this easy lemma.)

   (d) (**15** points) Extend the proof of the Progress Lemma to account for our additions. Include only the new cases. (Note you are only proving Progress, *not* Preservation or Substitution though those should also hold.) Hints:

   - You need to use the new case of Canonical Forms. Be clear about where you do so.
   - The strange evaluation rule will also be important. Be clear about where this is.

Solution:

(a) Rule GET-NONE is strange. It causes any program that tries to evaluate get None not to terminate. A real language would do something like throw an exception.

(b)

$$\frac{}{\Gamma \vdash \mathsf{None} : \tau\ \mathsf{option}} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \mathsf{Some}\ e : \tau\ \mathsf{option}} \qquad \frac{\Gamma \vdash e : \tau\ \mathsf{option}}{\Gamma \vdash \mathsf{get}\ e : \tau}$$

(c) If $\cdot \vdash v : \tau\ \mathsf{option}$, then v is None or there exists a v' such that v is Some v'.

(d) Recall the proof is by induction on the structure (syntax height) of e. New cases:

- If e is None, then e is a value.
- If e is Some e' for some e', then inverting the derivation of $\cdot \vdash e : \tau$ ensures $\cdot \vdash e' : \tau'$ for some τ'. So by induction either e' can take a step or it is a value. If e' can take a step, then e can take a step with SOME. If e' is a value, then e is a value.
- If e is get e' for some e', then inverting the derivation of $\cdot \vdash e : \tau$ ensures $\cdot \vdash e' : \tau\ \mathsf{option}$. So by induction either e' can take a step or it is a value. If e' can take a step, then e can take a step with GET-E. If e' is a value, then Canoncial Forms ensures it is None or Some v for some v. If it is None, we can take a step with GET-NONE (which is why we have our strange rule; in practice we would probably be stuck). If it is Some v, we can take a step with GET-SOME.

Name:____

(This page intentionally blank)