Name:_____

CSE505, Fall 2012, Midterm Examination
October 30, 2012

Rules:

- The exam is closed-book, closed-notes, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at Noon.**
- You can rip apart the pages if you like.
- There are **100 points** total, distributed **unevenly** among **5** questions (most of 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. (**26** points) This problem considers a language that is like the language for IMP *expressions* (not statements), but where we have *pixels* instead of *integers*. A pixel value contains three numbers between 0 and 255 (the first for red, the second for green, the third for blue, but that is not relevant much). Here is the syntax and an English description of the semantics:

$$\begin{array}{rcl} e & ::= & p \mid x \mid e + e \mid \mathsf{lighten}\ e \mid \mathsf{darken}\ e \\ p & ::= & \langle c, c, c \rangle \\ H & ::= & \cdot \mid H, x \mapsto p \\ (c & \in & \{0, 1, \ldots, 255\}) \\ (x & \in & \{\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{y}_1, \mathtt{y}_2, \ldots, \mathtt{z}_1, \mathtt{z}_2, \ldots, \ldots\}) \end{array}$$

- Heaps and variables work as usual, with values being pixels.
- As indicated in the syntax, all parts of pixel values must always be between 0 and 255 inclusive.
- Addition adds each component of its pixel arguments separately to produce a new pixel, with sums greater than 255 "rounded down" to 255.
- A "lighten" expression produces a pixel with each component being one more than it was in the argument, again with a max of 255 (i.e., 255 stays 255).
- A "darken" expression produces a pixel with each component being one less than it was in the argument, with a min of 0 (i.e., 0 stays 0).

(Notice *higher* values are *lighter*.)

(a) Give a large-step operational semantics for this language, with a judgment of the form $H\ ;\ e\ \Downarrow\ p$.
   - Use 5 rules. This is a good hint: there are other approaches that need many more rules.
   - Use from mathematics ("blue math" in terms of lecture) $\min(x, y)$ and $\max(x, y)$ for computing the minimum and maximum of two numbers. Also use addition and subtraction.

(b) Using your answer to part (a), *prove* this: $H\ ;\ \mathsf{lighten}\ e\ \Downarrow\ p$ if and only if $H\ ;\ e + \langle 1, 1, 1 \rangle\ \Downarrow\ p$.

(c) Define inference rules for a predicate $\mathsf{noblack}(e)$ that holds if none of the pixel constants in e are the constant $\langle 0, 0, 0 \rangle$ *and* e contains no "darken" expressions.

(d) *Disprove* this: If $H\ ;\ e\ \Downarrow\ p$ and $\mathsf{noblack(e)}$, then $p \neq \langle 0, 0, 0 \rangle$.

Solution:

(a)

$$\frac{}{H\ ;\ p\ \Downarrow\ p} \qquad \frac{}{H\ ;\ x\ \Downarrow\ H(x)}$$

$$\frac{H\ ;\ e_1\ \Downarrow\ \langle c_1, c_2, c_3 \rangle \qquad H\ ;\ e_2\ \Downarrow\ \langle c_4, c_5, c_6 \rangle}{H\ ;\ e_1 + e_2\ \Downarrow\ \langle \min(255, c_1 + c_4), \min(255, c_2 + c_5), \min(255, c_3 + c_6) \rangle}$$

$$\frac{H\ ;\ e\ \Downarrow\ \langle c_1, c_2, c_3 \rangle}{H\ ;\ \mathsf{lighten}\ e\ \Downarrow\ \langle \min(255, c_1 + 1), \min(255, c_2 + 1), \min(255, c_3 + 1) \rangle}$$

$$\frac{H\ ;\ e\ \Downarrow\ \langle c_1, c_2, c_3 \rangle}{H\ ;\ \mathsf{darken}\ e\ \Downarrow\ \langle \max(0, c_1 - 1), \max(0, c_2 - 1), \max(0, c_3 - 1) \rangle}$$

Name:_____

(More room for answering problem 1)

Solution:

(b) Prove the two directions separately. First assume $H \ ; \ \mathsf{lighten}\ e \ \Downarrow \ p$. Inversion (only the lighten rule applies) ensures there is some c_1, c_2, and c_3 such that p is $\langle \min(255, c_1 + 1), \min(255, c_2 + 1), \min(255, c_3 + 1) \rangle$ and $H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle$. So we can use $H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle$ and the addition rule to derive:

$$\frac{H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle \qquad \overline{H \ ; \ \langle 1, 1, 1 \rangle \ \Downarrow \ \langle 1, 1, 1 \rangle}}{H \ ; \ e + \langle 1, 1, 1 \rangle \ \Downarrow \ p}$$

Now assume $H \ ; \ e + \langle 1, 1, 1 \rangle \ \Downarrow \ p$. Then inversion (only the addition rule applies) ensures there is some c_1, c_2, and c_3 such that p is $\langle \min(255, c_1 + 1), \min(255, c_2 + 1), \min(255, c_3 + 1) \rangle$ and $H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle$ (because another inversion ensures $\langle 1, 1, 1 \rangle$ can evaluate only to itself). So we can use $H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle$ and the lighten rule to derive:

$$\frac{H \ ; \ e \ \Downarrow \ \langle c_1, c_2, c_3 \rangle}{H \ ; \ \mathsf{lighten}\ e \ \Downarrow \ p}$$

(c) (Note if you insist that $=$ and $\neq$ be used only on integers, then we need three rules for pixel constants.)

$$\frac{}{\mathsf{noblack}(x)} \qquad \frac{p \neq \langle 0, 0, 0 \rangle}{\mathsf{noblack}(p)} \qquad \frac{\mathsf{noblack}(e_1) \qquad \mathsf{noblack}(e_2)}{\mathsf{noblack}(e_1 + e_2)} \qquad \frac{\mathsf{noblack}(e)}{\mathsf{noblack}(\mathsf{lighten}\ e)}$$

(d) This is false because of variables. Consider any heap H where $H(x) = \langle 0, 0, 0 \rangle$. Then $H \ ; \ x \ \Downarrow \ \langle 0, 0, 0 \rangle$ but $\mathsf{noblack}(x)$.

Name:_____

2. (**19** points) In this problem we implement the language from Problem 1 in OCaml and write another related OCaml function. Take these type definitions as given:

```
type pixel = int * int * int
type exp = Pixel of pixel | Var of string | Add of exp * exp | Lighten of exp | Darken of exp
type heap = string -> pixel
```

(a) Write an OCaml function `interp` of type `heap -> exp -> pixel` that implements the semantics defined in Problem 1. Note that in addition to `+` and `-`, OCaml has functions `max` and `min` of type `int -> int -> int`.

(b) We call the first component of a pixel the red component. For example, in $\langle 4, 7, 9\rangle$, the red component is 4. Write an OCaml function `no_red_component` of type `exp -> exp` such that:

- For any heap `h` that contains only pixels with red components of zero, `interp h (no_red_component e)` never has any value returned from a call to `interp` (including recursive calls) contain a red component other than 0.
- Iff `interp h e` returns `(c1,c2,c3)`, then `interp h (no_red_component e)` returns `(0,c2,c3)` (i.e., the other components are the same).

Hint: Your function does *not* call `interp`. Rather, it translates its argument into a similar program that, when run, never produces a non-zero red component (assuming the heap has only pixels with red components of 0).

Solution:

(a)
```
let rec interp h e =
  match e with
    Pixel p -> p
  | Var s -> h s
  | Add(e1,e2)  -> let c1,c2,c3 = interp h e1 in
                   let c4,c5,c6 = interp h e2 in
                   (min 255 (c1+c4),min 255 (c2+c5),min 255 (c3+c6))
  | Lighten(e1) -> let c1,c2,c3 = interp h e1 in
                   (min 255 (c1+1),min 255 (c2+1),min 255 (c3+1))
  | Darken(e1)  -> let c1,c2,c3 = interp h e1 in
                   (max 0 (c1-1),max 0 (c2-1),max 0 (c3-1))
```

(b)
```
let rec no_red_component e =
  match e with
    Pixel(c1,c2,c3) -> Pixel(0,c2,c3)
  | Var _           -> e
  | Add(e1,e2)      -> Add(no_red_component e1, no_red_component e2)
  | Lighten e1      -> Add(no_red_component e1, Pixel(0,1,1))
  | Darken e1       -> Darken(no_red_component e1)
```

Name:_____

3. (**20** points) Take the IMP language we studied in class and remove if-statements and while-statements and the corresponding small-step rules. (We remove these only to make this problem shorter; it would still work with them.) Prove this theorem for the resulting language: If s does not contain the variable x and $H\ ;\ s\ \rightarrow\ H'\ ;\ s'$, then $H, x \mapsto c\ ;\ s\ \rightarrow\ H''\ ;\ s'$ for some H'' such that $H''(y) = H'(y)$ for all $y \neq x$.

   Assume this lemma is already proven, but clearly indicate where you use it: If e does not contain the variable x and $H\ ;\ e\ \Downarrow\ c'$, then $H, x \mapsto c\ ;\ e\ \Downarrow\ c'$.

   We expect a formal proof, but you do *not* need to define formally, "does not contain the variable x."

   Hint: You need induction, but you do not need a stronger induction hypothesis.

   Solution:
   The proof is by induction on the derivation of $H\ ;\ s\ \rightarrow\ H'\ ;\ s'$, with cases for the rule instantiated at the bottom of the derivation:

   - Assign: In this case, there is some x', e, and c' such that s is $x' := e$ (where by assumption $x' \neq x$ and e does not contain x), s' is skip, $H\ ;\ e\ \Downarrow\ c'$, and $H' = H, x' \mapsto c'$. By the lemma and $H\ ;\ e\ \Downarrow\ c'$, we know $H, x \mapsto c\ ;\ e\ \Downarrow\ c$, from which we can use Assign to derive $H, x \mapsto c\ ;\ x' := e\ \rightarrow\ H, x \mapsto c, x' \mapsto c'\ ;\ \mathsf{skip}$. This suffices because s' is skip and $H, x \mapsto c, x' \mapsto c'$ agrees with $H, x' \mapsto c'$ for any variable other than x.
   - Seq1: In this case, s is $\mathsf{skip}; s'$ and H' is H. Using Seq1 we can derive $H, x \mapsto c\ ;\ \mathsf{skip}; s'\ \rightarrow\ H, x \mapsto c\ ;\ s'$, which suffices because H and $H, x \mapsto c$ agree on all variables other than x.
   - Seq2: In this case, there is some s_1, s_1', and s_2 such that s is $s_1; s_2$, s' is $s_1'; s_2$, and $H\ ;\ s_1\ \rightarrow\ H'\ ;\ s_1'$ via a shorter derivation. Since $s_1; s_2$ does not contain x, neither does s_1. So by induction $H, x \mapsto c\ ;\ s_1\ \rightarrow\ H''\ ;\ s_1'$ and H' and H'' agree on all variables other than x. We can then use Seq2 to derive $H, x \mapsto c\ ;\ s_1; s_2\ \rightarrow\ H''\ ;\ s_1'; s_2$, which suffices.

Name:_____

4. (**15** points) **(10 for (a) and 5 for (b))** For this problem, you will provide an *encoding* for ML-style options (the type with constructors `None` and `Some`) in the call-by-value lambda-calculus.

   (a) Define these three lambda terms:
   - The "None" constructor "is" an option.
   - The "Some" constructor takes one argument and returns an option "containing" the argument.
   - The "use-it" constructor takes three (curried) arguments. If the first argument is "None" then it returns the second argument. Else it returns the result of calling the third argument with the value "contained in" the first argument.

   (b) Using your encoding and our left-to-right call-by-value small-step semantics, write out the sequence of lambda expressions by which "use-it" ("Some" v_1) v_2 $\lambda x.\ x$ becomes v_1.

   Note: Part (b) is helpful for checking part (a), but it is worth only 5 points, so you may wish to spend time on other problems instead.

Solution:
(There are an infinite number of correct solutions; here are two.)

(a)
- "None" $\lambda f.\ \lambda x.\ x$
- "Some" $\lambda y.\ \lambda f.\ \lambda x.\ f\ y$
- "use-it" $\lambda o.\ \lambda x.\ \lambda f.\ o\ f\ x$

(b)

$$
\begin{array}{ll}
 & (\lambda o.\ \lambda x.\ \lambda f.\ o\ f\ x)\ ((\lambda y.\ \lambda f.\ \lambda x.\ f\ y)\ v_1)\ v_2\ \lambda x.\ x \\
\to & (\lambda o.\ \lambda x.\ \lambda f.\ o\ f\ x)\ (\lambda f.\ \lambda x.\ f\ v_1)\ v_2\ \lambda x.\ x \\
\to & (\lambda x.\ \lambda f.\ (\lambda f.\ \lambda x.\ f\ v_1)\ f\ x)\ \ v_2\ \lambda x.\ x \\
\to & (\lambda f.\ (\lambda f.\ \lambda x.\ f\ v_1)\ f\ v_2)\ \lambda x.\ x \\
\to & ((\lambda f.\ \lambda x.\ f\ v_1)\ (\lambda x.\ x)\ v_2) \\
\to & (\lambda x.\ (\lambda x.\ x)\ v_1)\ v_2) \\
\to & (\lambda x.\ x)\ v_1 \\
\to & v_1
\end{array}
$$

(a)
- "None" $\lambda x.\ \lambda y.\ x$
- "Some" $\lambda z.\ \lambda x.\ \lambda y.\ y\ z$
- "use-it" $\lambda a.\ \lambda b.\ \lambda c.\ a\ b\ c$

(b)

$$
\begin{array}{ll}
 & (\lambda a.\ \lambda b.\ \lambda c.\ a\ b\ c)\ ((\lambda z.\ \lambda x.\ \lambda y.\ y\ z)\ v_1)\ v_2\ \lambda x.\ x \\
\to & (\lambda a.\ \lambda b.\ \lambda c.\ a\ b\ c)\ (\lambda x.\ \lambda y.\ y\ v_1)\ v_2\ \lambda x.\ x \\
\to & (\lambda b.\ \lambda c.\ (\lambda x.\ \lambda y.\ y\ v_1)\ b\ c)\ v_2\ \lambda x.\ x \\
\to & (\lambda c.\ (\lambda x.\ \lambda y.\ y\ v_1)\ v_2\ c)\ \lambda x.\ x \\
\to & (\lambda x.\ \lambda y.\ y\ v_1)\ v_2\ \lambda x.\ x \\
\to & (\lambda y.\ y\ v_1)\ \lambda x.\ x \\
\to & (\lambda x.\ x)\ v_1 \\
\to & v_1
\end{array}
$$

Name:_____

5. (**20** points)

   (a) We know the Preservation Theorem is true for the simply-typed lambda calculus. Is it still true if we add the rule $\frac{}{e \to 42}$ to the operational semantics? If so, briefly explain why. If not, provide a counterexample.

   (b) We know the Progress Theorem is true for the simply-typed lambda calculus. Is it still true if we add the rule $\frac{}{e \to 42}$ to the operational semantics? If so, briefly explain why. If not, provide a counterexample.

   (c) Suppose there is a different type system for lambda calculus with constants for which the Preservation Theorem is false. Will it still be false after adding the rule $\frac{}{e \to 42}$ to the operational semantics? Briefly explain your answer.

   (d) Suppose there is a different type system for lambda calculus with constants for which the Progress Theorem is false. Will it still be false after adding the rule $\frac{}{e \to 42}$ to the operational semantics? Briefly explain your answer.

Solution:

   (a) No. Consider, for example, $\lambda x.\ x \to 42$. Before the step we cannot give the term the type int, but after the step, this is the only type we can give.

   (b) Yes. Actually, Progress is now trivial because for any term we can step to 42. But we also can use all the other rules to take a step for non-value just as before adding this rule.

   (c) Yes, it will still be false. Whatever example violated preservation before will still be present after adding a rule to the operational semantics.

   (d) No, Progress is now true because for any e we can take a step to 42.