Please do not turn the page until everyone is ready.

Rules:

- The exam is open-book, open-note, closed electronics.
- Please stop promptly at 11:50.
- You can rip apart the pages, but please write your name on each page.
- You can turn in other pieces of paper.
- There are six questions (all with subparts), worth equal amounts. The subparts are not necessarily worth equal amounts.

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 roughly in the order we covered the material, not necessarily order of difficulty. Skip around.
- If you have questions, ask.
- Relax. You are here to learn, not beat the mean.
1. Consider this syntax for IMP expressions, which has (integer) division as the only arithmetic operator:

\[ e ::= c \mid x \mid e/e \]

(a) Give a large-step operational semantics of the form \( H ; e \Downarrow c \) for these expressions. Make sure that if evaluation of \( e \) under \( H \) would involve dividing by 0, then there is no \( c \) for which you can derive \( H ; e \Downarrow c \).

(Hint: You need 3 inference rules.)

(Note: You may assume \( H(x) \) is defined as in class.)

(b) Now suppose we add an explicit \textbf{error} result. Add inference rules to your previous answer so that \( H ; e \Downarrow v \) where \( v ::= c \mid \text{error} \). Make sure that if evaluation of \( e \) under \( H \) would involve dividing by 0, then \( H ; e \Downarrow \text{error} \).

(Hint: You need 3 more inference rules, so 6 total.)

(c) Does adding the rule \( H ; 0/e \Downarrow 0 \) change the semantics you defined for (a) and (b)? Explain.
2. Here is our unchanged syntax and semantics for IMP statements:

\[ s ::= \text{skip} \mid x := e \mid s ; s \mid \text{if } e \ s \ s \mid \text{while } e \ s \]

\[
\begin{align*}
\text{ASSIGN} & \\
H ; e \Downarrow c & \\
\hline
H ; x := e \rightarrow H, x \mapsto c ; \text{skip} & \\
\text{SEQ1} & \\
H ; \text{skip} ; s \rightarrow H ; s & \\
\text{SEQ2} & \\
H ; s_1 \rightarrow H' ; s_1' & \\
H ; s_1 ; s_2 \rightarrow H' ; s_1' ; s_2 & \\
\text{IF1} & \\
H ; e \Downarrow c & \\
c > 0 & \\
\hline
H ; \text{if } e \ s_1 \ s_2 \rightarrow H ; s_1 & \\
\text{IF2} & \\
H ; e \Downarrow c & \\
c \leq 0 & \\
\hline
H ; \text{if } e \ s_1 \ s_2 \rightarrow H ; s_2 & \\
\text{WHILE} & \\
H ; \text{while } e \ s \rightarrow H & \\
\text{while } e \ s \rightarrow H ; \text{if } e (s ; \text{while } e \ s) \text{ skip} & \\
\end{align*}
\]

(a) Define a judgment of the form \( \text{mysize}(s) = n \). Informally, \( n \) should be: (the number of \( \text{skip} \) statements in \( s \)) plus (two times the number of assignment statements in \( s \)). For example, you should be able to derive \( \text{mysize}(\text{skip} ; x := 0 ; y := 1) = 5 \).

(Hint: You need 5 inference rules.)

(b) Prove the following: If \( s \) has no while-statements or if-statements and \( H ; s \rightarrow H' ; s' \) and \( \text{mysize}(s) = n \) and \( \text{mysize}(s') = n' \), then \( n' < n \).

Note: This theorem is true with if-statements (but not while-statements), but you do not have to show this.

(c) Using part (b), argue informally (no proof required) that while-free programs terminate.
3. Describe what each of the following O'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 "hi" in
    let x = "mom" in
    g "pizza"

(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")
4. Consider a \( \lambda \)-calculus with \textit{pairs} built-in. That is, \((v_1, v_2)\) is a value if \(v_1\) and \(v_2\) are values, \((v_1, v_2).1 \to v_1\) and \((v_1, v_2).2 \to v_2\).

(a) Give an encoding of triples that uses pairs. You should define four terms: a three-argument function (using currying) to build a triple, and functions for returning the first, second, and third part of a triple. (By \textit{encoding}, we mean you may \textit{not} extend the syntax of the language.)

(b) In the simply-typed \( \lambda \)-calculus with pairs (and types of the form \( \tau_1 \ast \tau_2 \)), give \textit{two different types} that your function for forming a triple could have. (I.e., if \( e \) is your term for building a triple, give two \( \tau \) such that \( \vdash e : \tau \).)
5. Under what assumptions do the following terms type-check in the simply-typed \(\lambda\)-calculus? That is, for the given \(e\), describe all \(\Gamma\) and \(\tau\) such that \(\Gamma \vdash e : \tau\).

(a) \(e = x \ y\)
(b) \(e = \lambda x. (f (f \ x))\)
(c) \(e = \lambda x. (\lambda y. x)\)
(d) \(e = \lambda x. (x (\lambda y. x))\)
6. Recall how we extend the simply-typed \( \lambda \)-calculus with \( \text{fix} \):

\[
\begin{array}{ccc}
\frac{e \rightarrow e'}{\text{fix } e \rightarrow \text{fix } e'} & \frac{\Gamma \vdash e : \tau \rightarrow \tau}{\Gamma \vdash \text{fix } e : \tau} \\
\frac{\text{fix } \lambda x. \ e \rightarrow e[(\text{fix } \lambda x. \ e)/x]}{\Gamma \vdash \text{fix } \lambda x. \ e : \tau}
\end{array}
\]

Also we recall that this extension is type-safe.

(a) If we add the rule

\[
\Gamma \vdash \text{fix } e : \tau
\]

is our language still type-safe? If not, give a program that gets stuck. If so, argue the case of the Preservation Lemma proof for a typing derivation ending with this rule.

(b) If we add the rule

\[
\Gamma \vdash \text{fix } \lambda x. \ x : \tau
\]

is our language still type-safe? If not, give a program that gets stuck. If so, argue the case of the Preservation Lemma proof for a typing derivation ending with this rule.

Hint: The Preservation Lemma is: If \( \cdot \vdash e : \tau \) and \( e \rightarrow e' \), then \( \cdot \vdash e' : \tau \). We prove it by induction on the derivation of \( \cdot \vdash e : \tau \).