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:

\[ s ::= \text{skip} | x := e | s | \text{if } e \ s \ | \text{while } e \ s \]
\[ e ::= c | x | e + e | e * e \]
\( c \in \{\ldots, -2, -1, 0, 1, 2, \ldots\} \)
\( x \in \{x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots\} \)

\[
\frac{H ; e \downarrow c}{H_1 ; s_1 \rightarrow H_2 ; s_2}
\]

\[
\begin{array}{ccc}
\text{CONST} & \text{VAR} & \text{ADD} \\
H ; c \downarrow c & H ; x \downarrow H(x) & H ; e_1 \downarrow c_1 \ H ; e_2 \downarrow c_2 \\
\hline
\text{ASSIGN} & \text{SEQ1} & \text{SEQ2} \\
H ; e \downarrow c & H ; \text{skip} ; e \rightarrow H ; s & H ; s_1 \rightarrow H' ; s'_1 ; s_2 \\
\hline
\text{IF1} & \text{IF2} & \text{WHILE} \\
H ; \text{if } e \ s_1 \ s_2 \rightarrow H ; s_1 & H ; \text{if } e \ s_1 \ s_2 \rightarrow H ; s_2 & H ; \text{while } e \ s \rightarrow H ; \text{if } e \ (s ; \text{while } e \ s) \ \text{skip}
\end{array}
\]

\[
\begin{align*}
e & ::= \lambda x. e \mid x \mid e \mid e \mid c \\
v & ::= \lambda x. e \mid c \\
\tau & ::= \text{int} \mid \tau \rightarrow \tau
\end{align*}
\]

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

\[
\begin{align*}
e_1 & \rightarrow e'_1 & e_2 & \rightarrow e'_2 \\
e_1 \ e_2 & \rightarrow e'_1 \ e_2 & v \ e_2 & \rightarrow v \ e'_2
\end{align*}
\]

\[
\frac{e'[x] = e''}{x[e/x] = e}
\]

\[
\frac{y \not= x}{y[e/x] = y}
\]

\[
\frac{e_1[e/x] = e'_1}{(\lambda y. e_1)[e/x] = \lambda y. e'_1}
\]

\[
\frac{y \not= x \ y \not\in FV(e)}{e_1[e/x] = e'_1}
\]

\[
\frac{e_1[e/x] = e'_1}{(e_1 e_2)[e/x] = e'_1 e'_2}
\]

\[
\frac{e_1[e/x] = e'_1}{e_2[e/x] = e'_2}
\]

\[
\frac{\Gamma \vdash e : \tau}{\Gamma \vdash c : \text{int} \quad \Gamma \vdash x : \Gamma(x) \quad \Gamma, x : \tau_1 \vdash e : \tau_2 \\
\hline
\Gamma \vdash \lambda x. e : \tau_1 \rightarrow \tau_2 \quad \Gamma \vdash e_1 : \tau_2 \rightarrow \tau_1 \quad \Gamma \vdash e_2 : \tau_1}
\]

- Preservation: If \( \vdash e : \tau \) and \( e \rightarrow e' \), then \( \vdash e' : \tau \).
- Progress: If \( \vdash e : \tau \), then \( e \) is a value or there exists an \( e' \) such that \( e \rightarrow e' \).
- Substitution: If \( \Gamma, x : \tau' \vdash e : \tau \) and \( \vdash e' : \tau' \), then \( \Gamma \vdash e[e'/x] : \tau \).
1. In this problem, we change IMP by adding one more constant, □:

\[ c \in \{\ldots, -2, -1, 0, 1, 2, \ldots\} \cup \{\square\} \]

Because □ is a constant, it can also be an expression, the result of evaluating an expression, or the contents of a heap variable. However, □ is not a legal argument to any “math” operators like “(blue) plus” except “=” and “≠”.

Informally, if an expression has any subexpression that evaluates to □, then the expression evaluates to □.

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

(b) (17 points) Considering all the inference rules now in the language, prove that if \( e \) contains a □ and \( H ; e \Downarrow c \), then \( c \) is □. 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.
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:

\[
\begin{align*}
\text{SKIP} & : H; \text{skip} \downarrow H \\
\text{ASSIGN} & : H ; e \downarrow c \quad H; x := e \downarrow H, x \mapsto c \\
\text{SEQ} & : H; s_1 \downarrow H_1 \quad H; s_2 \downarrow H_2 \\
\text{IF-1} & : H; e \downarrow c \quad H; s_1 \downarrow H_1 \quad c > 0 \quad H; \text{if } e \; s_1 \; s_2 \downarrow H_1 \\
\text{IF-2} & : H; e \downarrow c \quad H; s_2 \downarrow H_2 \quad c \leq 0 \quad H; \text{if } e \; s_1 \; s_2 \downarrow H_2 \\
\text{WHILE} & : H; \text{while } e \; s \downarrow \text{skip} \downarrow H' \\
\end{align*}
\]

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.
Name: ________________________________

(This page intentionally blank)
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 ")
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.

\[
(\lambda p. p(\lambda x. \lambda y. x)) ((\lambda x. \lambda y. \lambda z. z \ x \ y) \ v_1 \ v_2)
\]

\[
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow \\
\rightarrow
\]

(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 \( (v_2, v_1) \).
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 ($e ::= ... | e - e$) and these rules:

\[
\begin{align*}
\frac{c_3 \text{ is math’s subtraction of } c_2 \text{ from } c_1}{c_1 - c_2 \rightarrow c_3} & \quad \frac{\Gamma \vdash e_1 : \text{int}}{\Gamma \vdash e_1 - e_2 : \text{int}}
\end{align*}
\]

(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.