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 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:

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

\[
H ; e \Downarrow c
\]

\[
\begin{array}{llll}
\text{CONST} & \quad \text{VAR} & \quad \text{ADD} & \quad \text{MULT} \\
H ; c \Downarrow c & H ; x \Downarrow H(x) & H ; e_1 \Downarrow c_1 \quad H ; e_2 \Downarrow c_2 & H ; e_1 \ast e_2 \Downarrow c_1 \ast c_2 \\
\hline
H_1 ; s_1 \rightarrow H_2 ; s_2 & H ; x := e \rightarrow H, x := c ; \text{skip} & H ; \text{skip} ; s \rightarrow H ; s & H ; s_1 \rightarrow H' ; s_1' \\
\hline
\end{array}
\]

\[
\begin{align*}
\text{IF1} &\quad H ; e \Downarrow c \quad c > 0 \\
\text{IF2} &\quad H ; e \Downarrow c \quad c \leq 0 \\
\text{WHILE} &\quad H ; \text{if } e \ s_1 \ s_2 \rightarrow H ; s_1 \\
&\quad H ; \text{while } e \ s \rightarrow H ; \text{if e (s; while e s)} \text{ skip}
\end{align*}
\]

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

\[
\begin{align*}
  e \rightarrow e'
\end{align*}
\]

\[
\begin{align*}
(\lambda x. \ e) \ v \rightarrow e[v/x] &\quad e_1 \rightarrow e'_1 &\quad e_2 \rightarrow e'_2 \\
\hline
\end{align*}
\]

\[
\begin{align*}
  x[e/x] = e &\quad e_1[e/x] = e'_1 &\quad y \neq x &\quad e_2[e/x] = e'_2 \\
\end{align*}
\]

\[
\begin{align*}
  (\lambda y. \ e_1)[e/x] = \lambda y. \ e'_1 \\
  e_1[e/x] = e'_1 &\quad e_2[e/x] = e'_2
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash e : \tau
\end{align*}
\]

\[
\begin{align*}
\Gamma \vdash c : \text{int} &\quad \Gamma \vdash x : \Gamma(x) &\quad \Gamma, x : \tau_1 \vdash e : \tau_2 &\quad \Gamma \vdash e_1 : \tau_1 \rightarrow \tau_1 &\quad \Gamma \vdash e_2 : \tau_2 \\
\end{align*}
\]

- If \( \vdash e : \tau \) and \( e \rightarrow e' \), then \( \vdash e' : \tau \).
- If \( \vdash e : \tau \), then \( e \) is a value or there exists an \( e' \) such that \( e \rightarrow e' \).
- If \( \Gamma, x : \tau' \vdash e : \tau \) and \( \Gamma \vdash e' : \tau' \), then \( \Gamma \vdash e[e'/x] : \tau \).
1. (IMP with booleans)
In this problem we extend the IMP expression language with booleans: true, false, negation, and inclusive-or. (Variables hold integers or booleans, but that is not directly relevant to the questions below.) The new syntax forms are:

\[
e ::= \ldots | \text{true} | \text{false} | \neg e | e \lor e
\]

The result of evaluating an expression can be an integer (not relevant below), true, or false. That is, we have \( H ; e \Downarrow v \) where \( v ::= c | \text{true} | \text{false} \).

Negation and inclusive-or can be “stuck” if a subexpression does not evaluate to a boolean.

(a) (10 points) Add rules to our large-step operational semantics to support the new syntax forms. For \( e_1 \lor e_2 \), use short-circuiting left-to-right evaluation (like || in many languages). If your rules all contain explicit uses of false and true, then you should expect to write 7 rules.

(b) (12 points) Theorem: If \( e \) always evaluates to a boolean, then \( e \) and \( \neg \neg e \) are equivalent.
- Restate this theorem formally.
- Prove this theorem formally.

(c) (10 points) Add implication \((e \Rightarrow e)\) to the language. Recall “\( a \) implies \( b \) if \( a \) is false or \( b \) is true.”
- Give large-step operational semantics rules that support this extension “directly,” using short-circuiting left-to-right evaluation. If your rules all contain explicit uses of false and true, then you should expect to write 3 rules.
- Give 1 rule that works just as well as your 3 rules by treating implication as a derived form. Remember this should be a large-step rule. Use \( v \) in this rule.
2. (18 points) (IMP with large-step semantics)

We can give IMP statements a large-step semantics with a judgment of the form \( H; s \downarrow H' \). The rules below do so, but there are errors. (The rules match neither our informal understanding nor our small-step semantics.) Find three errors (two of which are the same conceptual error), explain the problem, why it is a problem, and how to change the rules to solve the problem.

\[
\begin{array}{l}
\text{skip} & \quad \text{assign} & \quad \text{seq} \\
\hline
H; \text{skip} \downarrow H & H; e \downarrow c & H; s_1 \downarrow H_1 \quad H; s_2 \downarrow H_2 \\
\end{array}
\]

\[
\begin{array}{l}
\begin{array}{l}
\text{if1} \\
H; e \downarrow c \quad H; s_1 \downarrow H_1 \quad H; s_2 \downarrow H_2 \\
\hline
H; \text{if } e \quad s_1 \quad s_2 \downarrow H_1 \\
\end{array} \\
\begin{array}{l}
\text{if2} \\
H; e \downarrow c \quad H; s_1 \downarrow H_1 \quad H; s_2 \downarrow H_2 \\
\hline
H; \text{if } e \quad s_1 \quad s_2 \downarrow H_2 \\
\end{array} \\
\begin{array}{l}
\text{while} \\
H; (s; \text{while } e) \quad \text{skip} \downarrow H' \\
\hline
H; \text{while } e \quad s \downarrow H' \\
\end{array}
\]

Name: ________________________________

5
Consider this Caml code, which type-checks and runs correctly.

```caml
type dumbTree = Empty | Node of dumbTree * dumbTree

let rec s f t =
  match t with
  | Empty -> f t
  | Node(x,y) -> f t + s f x + s f y

let c1 t = s (fun x -> 1) t
let c2 t = s (fun x -> match x with Node(l,Empty) -> 1 | _ -> 0) t
```

(a) What are the types of \( s \), \( c1 \), and \( c2 \)?
(b) What do \( c1 \) and \( c2 \) compute? (Hint: The answers are straightforward.)
(c) Rewrite the last two lines of the code so they are shorter and equivalent.
4. (Coin-flipping in Lambda-Calculus)

In this problem we take the simply-typed lambda-calculus with conditionals (true, false, if $e_1 e_2 e_3$, and the type bool) and add a “coin-flip” expression, flip. This expression is not a value. Our call-by-value left-to-right small-step semantics has two new semantic rules:

$$
\text{flip} \rightarrow \text{true} \quad \text{flip} \rightarrow \text{false}
$$

(a) (5 points) In lambda-calculus with conditionals, write a (curried) function that returns the exclusive-or of its arguments. Do not use the constant true and use the constant false only once. (This does not require flip.)

(b) (5 points) Argue that for all $e$, $(\lambda x. e) \text{true}$ and $e[\text{true}/x]$ are equivalent under call-by-value.

(c) (8 points) Argue that depending on $e$, $(\lambda x. e) \text{flip}$ and $e[\text{flip}/x]$ may or may not be equivalent under call-by-value.

(d) (5 points) Give a typing rule for flip.

(e) (12 points) Assuming we have proofs of progress, preservation, and substitution for lambda-calculus with conditionals, explain how to extend the proofs for programs containing flip. Be clear about the induction hypothesis and what cases you are adding.