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.
For your reference:

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

\[
H; e \Downarrow c
\]

<table>
<thead>
<tr>
<th>CONST</th>
<th>VAR</th>
<th>ADD</th>
<th>MULT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H; c \Downarrow c)</td>
<td>(H; x \Downarrow H(x))</td>
<td>(H; e_1 \Downarrow c_1) (H; e_2 \Downarrow c_2)</td>
<td>(H; e_1 \Downarrow c_1) (H; e_2 \Downarrow c_2)</td>
</tr>
</tbody>
</table>

\[
H_1; s_1 \rightarrow H_2; s_2
\]

\[
\begin{align*}
  H; e \Downarrow c & \quad H; e \Downarrow c \\
  H; x := e \rightarrow H, x \rightarrow c; \text{skip} & \quad \text{skip}; s \rightarrow H; s \\
\end{align*}
\]

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

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

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

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

\[
\begin{align*}
  \Gamma \vdash e : \tau & \\
  \Gamma \vdash e : \tau & \\
  \Gamma \vdash e : \tau_1 & \quad \Gamma \vdash e : \tau_2 & \quad \Gamma \vdash e : \tau \quad \Gamma \vdash e : \tau_1 & \quad \Gamma \vdash e : \tau_2
\end{align*}
\]

- 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 \(\vdash e' : \tau'\), then \(\vdash e[e'/x] : \tau\).
1. In this problem, we consider an expression language that is like expressions in IMP except we remove multiplication and we add a \textit{global counter}. Our syntax is:

\[ e ::= c \mid x \mid e + e \mid \text{next} \]

Informally, the \textit{next} expression evaluates to the current counter-value and has the side-effect of incrementing the counter value.

(a) (11 points) Give a large-step semantics for this expression language. The judgment should have the form \( H; c_1; e \Downarrow c_2; c \) where:
- \( H, e, \) and \( c \) are like in IMP.
- \( c_1 \) is the value of the global counter before evaluation.
- \( c_2 \) is the value of the global counter after evaluation.

(b) (16 points) Prove this theorem: If \( H; c_1; e \Downarrow c_2; c \) and \( c_1' > c_1 \), then there exist \( c_2' \) and \( c' \) such that \( H; c_1'; e \Downarrow c_2'; c' \) and \( c_2' > c_2 \).

(c) (7 points) Suppose we also extend IMP statement semantics to support the global counter (so the judgment has the form \( H; c; s \rightarrow H'; c'; s' \)). Argue that this theorem is false: If \( H_1; c_1; s \rightarrow^* H_2; c_2; \text{skip} \) and \( c_1' > c_1 \), then there exist \( H_2' \) and \( c_2' \) such that \( H_1;c_1'; s \rightarrow^* H_2'; c_2'; \text{skip} \) and \( c_2' > c_2 \). You do not need to give the semantic rules for statements or show a full state sequence. Just give an example showing the theorem is false and explain why informally.
2. (10 points) In this problem we extend IMP statements with the construct \texttt{repeat } \textit{c} \texttt{ s}. Informally, the idea is to execute \textit{s} \textit{c} times. Here are two \textit{separate} ways one might add rules to the semantics:

- First way:

  \[
  \frac{c > 0}{H; \texttt{repeat } c \ s \rightarrow H; (s; \texttt{repeat } (c - 1) \ s)} \quad \frac{c \leq 0}{H; \texttt{repeat } c \ s \rightarrow H; \texttt{skip}}
  \]

- Second way:

\[
H; \texttt{repeat } c \ s \rightarrow H; (s; \texttt{if } (c - 1) (\texttt{repeat } (c - 1) \ s) \texttt{ skip})
\]

One of these ways is \textit{wrong} (in some situations) according to the informal description.

(a) Which way is wrong? Explain why it is wrong.

(b) Show how to change the wrong way to make it correct.
Name:______________________________________________________________

3. (18 points) Note there is a part (a) and part (b) to this problem.

(a) For each Caml function below (q1, q2, and q3):

- Describe in 1–2 English sentences what the function computes.
- Give the type of the function. (Hint: For all three functions, the type has one type variable.)

```caml
let q1 x =
  let rec g x y =
    match x with
    [] -> y
    | hd::tl -> g tl (hd::y)
  in g x []

let rec q2 f lst =
  match lst with
  [] -> []
  | hd::tl -> if f hd then hd::(q2 f tl) else q2 f tl

let q3 x g = g (g x)
```

(b) Consider this purposely complicated code that uses q3 as defined above.

```caml
let x = q3 2
let y z = z+z
let z = 9
let x = x y
```

After evaluating this code, what is x bound to?
4. In this problem, we consider a call-by-value lambda-calculus with very basic support for profiling: In addition to computing a value, it computes how many times an expression of the form \texttt{count } e \texttt{ is evaluated. Here is the syntax and operational semantics:}

\begin{align*}
e & ::= \lambda x. e | x | e \ e | c | \texttt{count } e
\end{align*}

\[ c; e \rightarrow c'; e' \]

\[
\frac{c; (\lambda x. e) \ v \rightarrow c; e[v/x]}{c; \texttt{count } v \rightarrow c + 1; v}
\]

\[
\frac{c; e_1 \rightarrow c'; e'_1}{c; e_2 \rightarrow c'; e'_2}
\]

\[
\frac{c; v \ e_2 \rightarrow c'; v \ e'_2}{c; \texttt{count } v \rightarrow c + 1; v}
\]

Given a source program \( e \), our initial state is \( 0; e \) (i.e., the count starts at 0). A program state \( c; e \) type-checks if \( e \) type-checks (i.e., the count can be anything).

(a) (6 points) Give a typing rule for \texttt{count } e \ that is sound and not unnecessarily restrictive.

(b) (13 points) State an appropriate Preservation Lemma for this language. Prove just the case(s) directly involving \texttt{count } e \ expressions.

(c) (13 points) State an appropriate Progress Lemma for this language. Prove just the case(s) directly involving \texttt{count } e \ expressions.

(d) (6 points) Give an example program that terminates in our language and would terminate if we changed function application to be call-by-name but under call-by-name it would produce a different resulting count. (Hint: This should not be difficult.)