You have **50 minutes** to complete this exam.

This is a **closed-notes** exam.

Do not share anything with other students. Do not talk to other students. Do not look other students’ exams. Do not expose your exam to easy viewing by other students. Violation of any of these rules will count as cheating.

If you believe there is an error or an ambiguous question, you may seek clarification from the instructor. Please speak quietly or write your question out.

Including this cover sheet and rules at the end, there are 7 pages to the exam, including one blank page for workspace. Once the exam begins, please verify that you have all 7 pages.

Please write your name and NetID in the spaces above.

Show your work. Partial credit will be given for incomplete answers.

If you finish with time remaining, check your work!
<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>100</strong></td>
<td></td>
</tr>
</tbody>
</table>
Problem 1. (25 points)
Suppose the type \texttt{arith} has been defined as follows:

\begin{verbatim}
Inductive arith :=
| Num (n : nat) 
| Plus (a : arith) (b : arith)
| Minus (a : arith) (b : arith).
\end{verbatim}

(a) (10 points) Write a function \texttt{plus\_count} that counts the number of \texttt{Plus} constructors in an \texttt{arith}. For instance, \texttt{plus\_count (Plus (Minus (Num 3) (Num 4)) (Plus (Num 5) (Num 6)))} should return 2.

\textbf{Solution:}

\begin{verbatim}
Fixpoint plus\_count (a : arith) : nat :=
match a with
| Num _ => 0
| Plus a b => S (plus\_count a + plus\_count b)
| Minus a b => plus\_count a + plus\_count b
end.
\end{verbatim}

(b) (15 points) Suppose you were given a function \texttt{size} that counts the number of constructors in an \texttt{arith}, and you wanted to prove that \texttt{plus\_count a < size a} for any \texttt{a}, by induction on \texttt{a}. For each case of the proof by induction, write out the goal for that case and any inductive hypotheses, either informally or formally. You only have to state the cases, not prove them.

\textbf{Solution:}

\begin{verbatim}
Num case: No hypotheses. Goal: plus\_count (Num n) < size (Num n).

Plus case: Hypotheses: \texttt{plus\_count a < size a} and \texttt{plus\_count b < size b}.
Goal: plus\_count (Plus a b) < size (Plus a b).

Minus case: Hypotheses: \texttt{plus\_count a < size a} and \texttt{plus\_count b < size b}.
Goal: plus\_count (Minus a b) < size (Plus a b).
\end{verbatim}
Problem 2. (20 points)
Suppose we wanted to write a function `apply_each` that takes a list, each of whose elements is a pair 
\((f, x)\) of a function and an argument, and returns the list formed by applying each function \(f\) to its 
associated argument \(x\). For instance, 
\[
\text{apply} \_ \text{each} \ [(\text{fun } x \Rightarrow x + 1, 3); (\text{pred}, 6); (\text{fun } y \Rightarrow y, 1)] \]
should return \([4; 5; 1]\).

(a) (8 points) What should the type of `apply_each` be?

Solution:

\[
\text{forall } A \text{ B, list } ((A \rightarrow B) \ast A) \rightarrow list \text{ B}
\]

(b) (12 points) Write the function `apply_each`. For full credit, do not use built-in list functions 
like `map` or `fold`.

Solution:

\[
\text{Fixpoint apply}_\text{each} \ l :=
\quad \text{match } l \text{ with}
\quad \mid \ [ ] \Rightarrow [ ]
\quad \mid (f, x) :: \text{rest} \Rightarrow f \ x :: \text{apply}_\text{each} \ \text{rest}
\quad \text{end.}
\]
Problem 3. (24 points)
Consider the following proof state:

\[ P, Q, R : \text{Prop} \]
\[ H : P \lor Q \]
\[ H_0 : \text{False} \]
\[ H_1 : \forall P, Q : \text{Prop}, P \Rightarrow (P \Rightarrow Q) \Rightarrow P \land Q \]

\[ P \land R \]

Describe the effect each of the following tactics will have on the proof state, mentioning any changes to the goal(s) and/or the premises. If the tactic does not apply, write “error”.

(a) (4 points) split

**Solution:** Goal becomes two goals, with conclusions \( P \) and \( R \).

(b) (4 points) left

**Solution:** error

(c) (4 points) destruct \( H \) as \([H | H]\)

**Solution:** Goal becomes two goals, with hypothesis \( H \) becoming \( P \) in the first goal and \( Q \) in the second goal.

(d) (4 points) destruct \( H_0 \)

**Solution:** Goal is solved.

(e) (4 points) apply \( H_1 \)

**Solution:** Goal becomes two goals, with conclusions \( P \) and \( P \Rightarrow R \).

(f) (4 points) rewrite \( H_1 \)

**Solution:** error
Problem 4. (31 points)
A list of \texttt{nats} is \texttt{sorted} if each element is less than or equal to the following element.

(a) (12 points) Write an inductive relation \texttt{sorted} encoding this property. You may find it easiest to have one case for the empty list, one for one-element lists, and one for lists of two or more elements.

\begin{verbatim}
Solution:
Inductive sorted : list nat -> Prop :=
| sorted_nil : sorted []
| sorted_single a : sorted [a]
| sorted_cons : forall a b l (Hle : a <= b) (Hrest : sorted (b :: l)),
  sorted (a :: b :: l).
\end{verbatim}

(b) (19 points) Using your definition, write a detailed informal proof of the following fact: if two lists $l_1$ and $l_2$ are both sorted, and the last element of $l_1$ is less than or equal to the first element of $l_2$, then $l_1 \ ++ \ l_2$ is sorted. When you use a case of \texttt{sorted}, mention it by name. Hint: this is easiest to prove by induction on the fact that $l_1$ is sorted (i.e., on the hypothesis that says \texttt{sorted} $l_1$).

\begin{verbatim}
Solution: By induction on the fact that $l_1$ is sorted. There are three cases, one for each constructor of \texttt{sorted}:

1. $l_1$ is $\[]$. Then $l_1 \ ++ \ l_2$ is $\[] \ ++ \ l_2$, which is sorted by assumption.

2. $l_1$ is $[a]$ for some $a$. Then $l_1 \ ++ \ l_2$ is $a :: l_2$. There are two possible cases for $l_2$:
   \begin{itemize}
   \item $12$ is $\[]$. Then $11 \ ++ \ 12$ is $\[] \ ++ \ 12$, which is sorted by \texttt{sorted_single}.
   \item $12$ is $b :: l$ for some $b$ and $l$. Then $11 \ ++ \ 12$ is $a :: b :: l$. Furthermore, we know that \texttt{sorted} $(b :: l)$ by assumption, and since the last element of $11$ (which is $a$) is less than or equal to the first element of $12$ (which is $b$), we know that $a <= b$. So we can conclude that $11 \ ++ \ 12$ is sorted by \texttt{sorted_con}
   \end{itemize}

3. $l_1$ is $a :: b :: l$, where $a <= b$ and \texttt{sorted} $(b :: l)$, and the inductive hypothesis tells us that if the last element of $b :: l$ is less than or equal to the first element of $12$, then $b :: l \ ++ \ 12$ is sorted. The last element of $b :: l$ is the same as the last element of $a :: b :: l$ (that is, $11$), so we know it’s less than or equal to the first element of $12$. Thus, by the inductive hypothesis, $b :: l \ ++ \ 12$ is sorted, and so by \texttt{sorted_con} we know that $a :: b :: l \ ++ \ 12$ is sorted too. QED.
\end{verbatim}
1 Scratch Space