The following exercises are typical exam exercises. You are supposed to solve them on a sheet of paper, without using Isabelle/HOL.

**Exercise 14.1  Inductive Predicates**

Consider the following inductive predicate, which characterizes odd natural numbers.

\[ \text{inductive} \quad \text{odd} :: \nat \Rightarrow \bool \]  
\[ \quad \text{where} \]
\[ \quad \text{Suc} \_ \text{Suc} : \text{odd} \ (\text{Suc} \ (\text{Suc} \ n)) \Rightarrow \text{odd} \ (\text{Suc} \ (\text{Suc} \ n)) \]

Using the induction principle for the predicate \text{odd}, it can be proven that three times any odd number is also odd:

\[ \text{lemma} \quad \text{odd} \ n \Rightarrow \text{odd} \ (n + n + n) \]

\[ \text{proof} \quad (\text{induct rule: odd.induct}) \]
\[ \quad \text{case} \ \text{Suc} \_ \text{Suc} \ n \ \text{show} \ ?\text{case} \ \text{by} \ (\text{simp add: odd.intros}) \]
\[ \quad \text{next} \]
\[ \quad \text{case} \ (\text{Suc} \_ \text{Suc} \ n) \ \text{thus} \ ?\text{case} \ \text{by} \ (\text{simp add: odd.intros}) \]
\[ \text{qed} \]

First, write down precisely what subgoals remain after performing induction. How many cases are there? Which assumptions are available, and what conclusion must be proved in each case? Next, describe how each case can be proved. Which simplification rules or introduction rules are used to prove each case?

**Exercise 14.2  Collecting Semantics**

Recall the datatype of annotated commands (type `'a acom`) and the collecting semantics (function \text{step} :: state set \Rightarrow state set acom \Rightarrow state set acom) from the lecture. We reproduce the definition of \text{step} here for easy reference. (Recall that \text{post} \ c simply returns the right-most annotation from command \ c.)
Exercise 14.3 Substitution

**type synonym** vname = "char list"

**datatype** aexp = N int | V vname | Plus aexp aexp

**fun** subst::"aexp ⇒ vname ⇒ aexp ⇒ aexp" where

"subst (N i) v a' = (N i)"

| "subst (V v') v a' = (if a=v' then a' else (V v'))" |
| "subst (Plus a1 a2) v a' = Plus (subst a1 v a') (subst a2 v a')" |

**fun** occurs::"aexp ⇒ vname ⇒ bool" where

"occurs (N _) _ ←→ False"

| "occurs (V v) v ←→ v=v'" |
| "occurs (Plus a1 a2) v ←→ occurs a1 v ∨ occurs a2 v"
lemma lem1: “¬occurs a v → subst a v a' = a”
  by (induct a) auto

lemma lem2: “occurs (subst (V v) v (V v)) v” by simp