Isabelle/HOL Exercises
Lists

```
consts
  occurs :: "'a ⇒ 'a list ⇒ nat"
primrec
  "occurs a [] = 0"
  "occurs a (x#xs) = (if (x=a) then Suc(occurs a xs) else occurs a xs)"

lemma [simp]:"occurs a (xs @ ys) = occurs a xs + occurs a ys "
  apply (induct_tac xs)
  apply auto
  done

lemma "occurs a xs = occurs a (rev xs)"
  apply (induct_tac xs)
  apply auto
  done

lemma "occurs a xs <= length xs"
  apply (induct_tac xs)
  apply auto
  done

lemma "occurs a (replicate n a) = n"
  apply (induct_tac n)
  apply auto
  done

consts
  areAll :: "'a list ⇒ 'a ⇒ bool"
primrec
  "areAll [] a = True"
  "areAll (x#xs) a = ((x=a) ∧ (areAll xs a))"

lemma "areAll xs a ⟶ occurs a xs = length xs"
  apply (induct_tac xs)
  apply auto
  done
```

```
lemma "occurs a xs = length xs ⟶ areAll xs a"
```
— additional lemmas needed
```
  apply (induct_tac xs)
    apply auto
```
⋮
```
   lemma l:"occurs a xs < Suc( length xs)"
     apply (induct_tac xs)
     apply auto
     done

   lemma a:"(a::nat) < b ⟶ (a ~= b)"
     apply auto
     done

   lemma ne:"occurs a xs ~= Suc( length xs)"
     apply (auto simp:l a)
     done

   lemma "occurs a xs = length xs ⟶ areAll xs a"
     apply (induct_tac "xs")
     apply (auto simp:ne)
     done

consts
  delall :: "'a ⇒ 'a list ⇒ 'a list"
primrec
  "delall a [] = []"
  "delall a (x#xs) = (if (x=a) then (delall a xs) else (x#delall a xs))"


lemma "occurs a (delall a xs) = 0"
  apply (induct_tac xs)
  apply auto
  done

consts
  del1 :: "'a ⇒ 'a list ⇒ 'a list"
primrec
  "del1 a [] = []"
  "del1 a (x#xs) = (if (x=a) then xs else (x#del1 a xs))"
```

lemma *"Suc (occurs a (del1 a xs)) = occurs a xs"*
— wrong; precondition needed
⋮

lemma *"xs ~= [] ⟶ Suc (occurs a (del1 a xs)) = occurs a xs"*
apply *(induct_tac xs)*
apply *auto*
— still wrong
⋮

lemma *"0 < occurs a xs ⟶ Suc (occurs a (del1 a xs)) = occurs a xs"*
apply *(induct_tac xs)*
apply *auto*
— correct!
done

consts
replace :: "'a ⇒ 'a ⇒ 'a list ⇒ 'a list"
primrec
"replace a b [] = []"
"replace a b (x#xs) = (if (x=a) then (b#(replace a b xs))
else (x#(replace a b xs)))"

lemma *"occurs a xs = occurs b (replace a b xs)"*
apply *(induct_tac xs)*
apply *auto*
— wrong; precondition needed
⋮

lemma *"occurs b xs = 0 ∨ a=b ⟶ occurs a xs = occurs b (replace a b xs)"*
apply *(induct_tac xs)*
apply *auto*
done

consts
remDups :: "'a list ⇒ 'a list"
primrec
"remDups [] = []"
"remDups (x#xs) = (if (0 < occurs x xs) then (remDups xs)
else (x#(remDups xs)))"

```
lemma h:"occurs x xs = 0 ⟶ occurs x (remDups xs) = 0"
  apply (induct_tac xs)
  apply auto
  done

lemma "occurs x (remDups xs) <= 1"
  apply (induct_tac xs)
  apply (auto simp:h)
  done

consts
  unique :: "'a list ⇒ bool"
primrec
  "unique [] = True"
  "unique (x#xs) = (if (occurs x xs = 0) then (unique xs) else False)"

lemma "unique(remDups xs)"
  apply (induct_tac xs)
  apply (auto simp:h)
  done

end
```