In each of the exercises, prove the proposition in Isabelle/Isar using the inference rules described in the lecture slides and the following rules.

**Equal-Intro-Forwards**

If \( L_i \) proves \( P \rightarrow Q \) and \( L_j \) proves \( Q \rightarrow P \), then write

\[
\text{from } L_i \ L_j \text{ have } L_k: \ "P = Q" \quad \text{by blast}
\]

**Equal-Intro-Backwards**

have \( L_k: \ "P = Q" \)

proof

assume \( L_i: \ "P" \)

\[ \vdots \]

\[ \ldots \text{show } "Q" \ldots \]

next

assume \( L_i: \ "Q" \)

\[ \vdots \]

\[ \ldots \text{show } "P" \ldots \]

qed

**Equal-Elim (1)**

If \( L_i \) proves \( P = Q \) and \( L_j \) proves \( Q \), then write

\[
\text{from } L_i \ L_j \text{ have } L_k: \ "P" \ldots
\]

**Equal-Elim (2)**

If \( L_i \) proves \( P = Q \) and \( L_j \) proves \( P \), then write

\[
\text{from } L_i \ L_j \text{ have } L_k: \ "Q" \ldots
\]
You may also use the law of excluded middle:

**have** \( L_k: \neg P \lor P \) by (rule excluded_middle)

**Exercise 1** Modus tollens: \( \neg Q \land (P \rightarrow Q) \rightarrow \neg P \).

**Exercise 2** Disjunctive syllogism: \( (P \lor Q) \land \neg P \rightarrow Q \).

**Exercise 3** Resolution: \( (p \lor q) \land (\neg p \lor r) \rightarrow q \lor r \).

**Exercise 4** Equivalence of implication and disjunction with negation: \( (P \rightarrow Q) = (\neg P \lor Q) \).

**Exercise 5** \((A \land B) = (B \land C) \lor B \rightarrow A = C\).

**Exercise 6** Prove the universal modus ponens rule. The statement of the theorem is shown below. (Replace the oops with your proof.) You can give a name to a theorem by putting that name after the **theorem** keyword followed by a colon. The **assumes/shows** notation allows you to label the assumptions and separate them out from the conclusion.

**theorem** universal_modus_ponens:
  **assumes** 1: "\( \forall x. P(x) \rightarrow Q(x) \)"
  and 2: "P(a)"
  **shows** "Q(a)"
  oops

**Exercise 7** Use the universal modus ponens theorem to shorten the proof (from the lecture slides) that every man has two legs. You can apply a theorem just like a rule, write by (rule universal_modus_ponens).

**Exercise 8** Prove that if \( \exists x. P \land Q x \), then \( P \land (\exists x. Q x) \). Note that the notation \( Q x \) is equivalent to \( Q(x) \).

For the next few exercises, use the following definitions for even and odd.

**definition** even :: "nat \Rightarrow bool" where
  "even n \equiv \exists m. n = 2 \times m"

**definition** odd :: "nat \Rightarrow bool" where
  "odd n \equiv \exists m. n = 2 \times m + 1"

**Exercise 9** Prove that zero is not odd.

**Exercise 10** Prove that the sum of two odd natural numbers is even.

**Exercise 11** Prove that every odd natural number is the difference of two squares.
theorem modus_tollens: "¬ q ∧ (p → q) → ¬ p"
proof
  assume 1: "¬ q ∧ (p → q)"
  from 1 have 2: "¬ q" ..
  from 1 have 3: "p → q" ..
  show "¬ p"
  proof
    assume 4: "p"
    from 3 4 have 5: "q" ..
    from 2 5 show "False" ..
  qed
qed

theorem disjunctive_syllogism: "(p ∨ q) ∧ ¬ p → q"
proof
  assume 1: "(p ∨ q) ∧ ¬ p"
  from 1 have 2: "¬ p" ..
  from 1 have 4: "p ∨ q" ..
  note 4
  moreover {
    assume 3: "p"
    from 2 3 have "q" ..
  } moreover {
    assume 1: "q"
    from 1 have "q".
  } ultimately show "q" ..
qed

theorem resolution: "(p ∨ q) ∧ (¬ p ∨ r) → q ∨ r"
proof
  assume 1: "(p ∨ q) ∧ (¬ p ∨ r)"
  from 1 have "p ∨ q" ..
  moreover {
    assume 3: "p"
    from 1 have "¬ p ∨ r" ..
    moreover { assume 4: "¬ p"
      from 4 3 have "q ∨ r" ..
    } moreover {
      assume 5: "r"
      from 5 have "q ∨ r" ..
    } ultimately have "q ∨ r" ..
  } moreover {
    assume 6: "q"
    from 6 have "q ∨ r" ..
  } ultimately show "q ∨ r" ..
qed

theorem imp_disj_equiv: "(P → Q) = (¬ P ∨ Q)"
proof
  assume 1: "P → Q"

have 4: "¬ P ∨ P" by (rule excluded_middle)

note 4

moreover { 
  assume 2: "¬ P"
  from 2 have "¬ P ∨ Q" ..
} moreover {
  assume 2: "P"
  from 1 2 have 3: "Q" ..
  from 3 have "¬ P ∨ Q" ..
} ultimately show "¬ P ∨ Q" ..

next

assume 1: "¬ P ∨ Q"
show "P → Q"

proof
  assume 2: "P"
  note 1
  moreover {
    assume 3: "¬ P"
    from 3 2 have "Q" ..
  }
  moreover {
    assume 3: "Q"
    from 3 have "Q" .
  }
  ultimately show "Q" ..

qed

qed

gs

gs

gs

gs

theorem "((A ∧ B) = (B ∧ C)) ∧ B → A = C"

proof
  assume 1: "(A ∧ B) = (B ∧ C) ∧ B"
  from 1 have 2: "(A ∧ B) = (B ∧ C)" ..
  from 1 have 3: "B" ..
  show "A = C"
  proof
    assume 4: "A"
    from 4 3 have 5: "A ∧ B" ..
    from 2 5 have 6: "B ∧ C" ..
    from 6 show "C" ..

next
  assume 4: "C"
  from 3 4 have 4: "B ∧ C" ..
  from 2 4 have 5: "A ∧ B" ..
  from 5 show "A" ..

qed

qed

theorem universal_modus_ponens:
  assumes 1: "∀ x. P(x) → Q(x)"
  and 2: "P(a)"
  shows "Q(a)"
  proof
    from 1 have 3: "P(a) → Q(a)" ..
from 3 2 show "Q(a)" ..
qed

theorem
assumes 1: "∀ x. man(x) −→ human(x)"
and 2: "∀ x. human(x) −→ hastwolegs(x)"
shows "∀ x. man(x) −→ hastwolegs(x)"

proof
  fix m
  show "man(m) −→ hastwolegs(m)"
  proof
    assume 3: "man(m)"
    from 1 3 have 5: "human(m)" by (rule universal_modus_ponens)
    from 2 5 show "hastwolegs(m)" by (rule universal_modus_ponens)
  qed
qed

theorem
assumes 1: "∃ x. P ∧ Q(x)"
shows "P ∧ (∃ x. Q(x))"

proof
  from 1 obtain y where 2: "P ∧ Q(y)" ..
  from 2 have 3: "P" ..
  from 2 have 4: "Q(y)" ..
  from 4 have 5: "∃ x. Q(x)" ..
  from 3 5 show "P ∧ (∃ x. Q(x))" ..
qed

theorem zero_not_odd: "¬ odd 0"

proof
  assume 1: "odd 0"
  from 1 have 2: "∃ m::nat. 0 = 2 * m + 1" unfolding odd_def .
  from 2 obtain k::nat where 3: "0 = 2 * k + 1" ..
  from 3 show "False" by simp
qed

theorem
assumes 1: "odd n" and 2: "odd m"
shows "even (n + m)"

proof
  from 1 have "∃ m::nat. n = 2 * m + 1" unfolding odd_def .
  from this obtain k where 3: "n = 2 * k + 1" ..
  from 2 have "∃ q::nat. m = 2 * q + 1" unfolding odd_def .
  from this obtain q where 4: "m = 2 * q + 1" ..
  from 3 4 have "n + m = 2 * k + 2 * q + 2" by simp
  also have "... = 2 * (k + q + 1)" by simp
  finally have 5: "n + m = 2 * (k + q + 1)" .
  from 5 have 6: "∃ x::nat. (n + m) = 2 * x" ..
  from 6 show "even (n + m)" unfolding even_def .
qed
theorem
  assumes 1: "odd n"
  shows "∃ x. ∃ y. n = x * x - y * y"
proof -
  from 1 have "∃ m. n = 2 * m + 1" unfolding odd_def .
  from this obtain k where 2: "n = 2 * k + 1" ..
  from 2 have 3: "n = (k + 1) * (k + 1) - k * k" by simp
  from 3 have 4: "∃ y. n = (k + 1) * (k + 1) - y * y" ..
  from 4 show "∃ x. ∃ y. n = x * x - y * y" ..
qed