1. Give three λ-terms M such that $\vdash M : \tau \to \tau \to \tau$, for τ an arbitrary type.

   We count only terms up to renaming of variables; so $\lambda xy.xy$ and $\lambda yx.yx$ are considered the same (as they can be obtained by renaming x into y and vice versa), but different from $\lambda xy.yx$. Is this a reasonable way to count terms inhabiting types, i.e. to count proofs of propositions, vis-à-vis the Curry–Howard isomorphism?

2. Bonus: Show that, among the three λ-terms in the previous exercise, at least two must be $=_\beta$-related.

3. Reduce the λ-term $M = (\lambda x.xx)(\lambda yz.yz)$ to normal form N (it requires 3 $\to_\beta$-steps; give each of them).

4. Show that there is no λ-term M such that $\vdash M : (\tau \to \tau) \to \tau$, for τ an arbitrary type.
   Hint: normalization or Kripke models (1 cross each)

5. Compute the translation $(\mathsf{SS(KI)})_\lambda$, i.e. the translation of W on slide 21 of the previous week, and reduce the resulting λ-term to normal form.

6. Bonus (1 cross per item): The Haskell expression `(\x -> \x -> x)` corresponds to the λ-term $\lambda x.\lambda x.x$. Asking Haskell the type of the former yields `p1 -> p2 -> p2`.
   - Explain why using the type assignment rules for λ-calculus (slide 13), we can infer $\vdash \lambda y.\lambda x.x : \sigma \to (\tau \to \tau)$, but not $\vdash \lambda x.\lambda x.x : \sigma \to (\tau \to \tau)$, assuming $\sigma \neq \tau$ (cf. the remark there).
     Could one overcome this limitation? That is, could our type assignment system be adapted such that we do have $\vdash \lambda x.\lambda x.x : \sigma \to (\tau \to \tau)$?
   - Just as per our conventions $\lambda yx.x$ is shorthand for $\lambda y.\lambda x.x$, in Haskell `(\y x -> x)` is shorthand for `(\y -> \x -> x)` Do `(\x x -> x)` and `(\x -> \x -> x)` have the same type in Haskell? Can you explain why (not)?

7. SC is defined (slide 17) in terms of itself: SC of M is defined in terms of SC of $\vec{N}$. Argue that SC is still well-defined, i.e. that it is a proper inductive definition for simply typed terms (either CL-terms or λ-terms).

8. Work out the details of the first item, the variable case, (slide 17) of the proof that all simply typed CL terms are SC. In particular, show that SC entails SN and that every typed variable is SC.

9. Complete the details of the proof (slide 20) that inhabitation is decidable. More precisely, give both the details of the proof of the subformula property, and of that the subformula property entails decidability of inhabitation.

10. Bonus (worth 4 crosses): implement an inhabitation checker for implicational intuitionistic logic, based on the previous exercise.

11. Bonus: Suppose combinatory logic would have another constant J having some type and reduction rule, for all terms M_i, $JM_1 \dots M_n \to_w E$ where E is an expression only constructed from applications and the M_i and both sides have the same base type, e.g. $\Gamma \vdash \mathsf{J} : (\sigma \to \sigma \to \tau) \to \sigma \to \tau$ with rule $\mathsf{J}M_1 M_2 \to_w M_1 M_2 M_2$ (note the conditions hold for K and S). Is $\to_w$ still strongly normalizing? If it is not, give a specific J and rule for it allowing an infinite reduction. If it is, give a proof.

12. Bonus (worth 3 crosses): directly show weak normalization of $\to_\beta$ on typable λ-terms, without showing (a property that entails) strong normalization . Hint: an idea analogous to that for the cut elimination procedure in the book works, judiciously choosing a $\to_\beta$ step among the possible ones and showing that that decreases some measure.

13 Bonus (worth 4 crosses): Prove or disprove that the tableau cut-elimination procedure in Fitting is strongly normalizing. Proceed as follows: the proof of Lemma 8.9.3 establishes weak normalization by transforming minimal cuts. (Try to) verify whether

- correctness of the transformations depends on minimality,
- the induction (see p. 232) used in the proof works for non-minimal cuts,
- strong normalization, when eliminating arbitrary cuts, holds.