1. For each type, construct a simply-typed lambda calculus term (variables, functions, and function application only) whose most general type is that type, or argue that no term has that type. (Hint: You can double-check your answers in OCaml.)

(a) $\alpha \to \beta \to \beta$
(b) $(\alpha \to \beta \to \gamma) \to \beta \to \alpha \to \gamma$
(c) $\alpha \to \beta$
(d) $\alpha \to \alpha \to \alpha$

2. Does the simply-typed lambda calculus with integers have a subject expansion property, meaning if $\Gamma \vdash e : \tau$ and $e' \to e$, does $\Gamma \vdash e' : \tau$? Here $\to$ is reduction under call-by-value semantics. Either prove that subject expansion holds, or give a counterexample showing that it does not hold.

3. Suppose we were to add booleans to the simply-typed lambda calculus:

$$e ::= x \mid n \mid true \mid false \mid \lambda x.e \mid e \ e \mid if\ e\ then\ e\ else\ e$$

(a) Write down small-step call-by-value semantic rules for the new forms $true$, $false$, and $if$. (Here $if$ should behave as it does in O’Caml, evaluating to the result of either the true or false branch depending on the guard.)

(b) Extend the typing judgment $\Gamma \vdash e : \tau$ to the new forms $true$, $false$, and $if$.

(c) Prove progress and preservation for the extended language. (You don’t need to reprove the cases for the old forms.)