1. Consider the following version of the Simply Typed λ-Calculus, where we only have one base type `nat`:

\[
\begin{align*}
\tau & ::= \text{nat} | \tau \rightarrow \tau & \text{types} \\
v & ::= x | n | \lambda x : \tau . e & \text{values} \\
e & ::= v | \text{succ}(e) | \text{pred}(e) | \text{if0} e \text{ then } e \text{ else } e | e \ e & \text{expressions}
\end{align*}
\]

The typing rules and evaluation rules are the same as we have discussed in class—with the minor exception of `if0`. Here are the four rules of interest for `if0`—one typing rule and three rules for the small-step semantics:

\[
\begin{align*}
\Gamma \vdash e_1 : \text{nat} & \quad \Gamma \vdash e_2 : \tau & \quad \Gamma \vdash e_3 : \tau \\
\Gamma \vdash \text{if0} e_1 \text{ then } e_2 \text{ else } e_3 : \tau & \\
& \quad e_1 \mapsto e'_1 \\
& \quad \text{if0} e_1 \text{ then } e_2 \text{ else } e_3 \mapsto e'_1 \text{ if0} e'_1 \text{ then } e_2 \text{ else } e_3 & \\
& \quad e_1 \neq 0 \\
& \quad \text{if0} 0 \text{ then } e_2 \text{ else } e_3 \mapsto e_2 & \quad \text{if0} n \text{ then } e_2 \text{ else } e_3 \mapsto e_3
\end{align*}
\]

Questions:

(a) Define the substitution function \( \{v/x\}e \) where \( v \) is assumed to be closed.

(b) Formally state the substitution lemma for this language. (Informally: Substituting a value of the “right” type into a well-typed expression keeps that expression well-typed.)

(c) Prove your substitution lemma.

2. Canonical forms and progress:

(a) For the same language as in question 1, state and prove the canonical forms lemma. (See Lemma 10.2 on page 61 of the textbook for reference. Your proof should be spelled out in more detail than the one-liner in the textbook.)

(b) State the progress lemma for this language.
(c) Show the proof of the progress lemma for the case of application \((e_1 e_2)\).

3. Consider the following alternative language design:

\[
\begin{align*}
\tau & ::= \text{nat} | \tau \to \tau & \text{types} \\
v & ::= x | n | \lambda x : \tau. e & \text{values} \\
e & ::= v | \text{if0} e \text{ then } e \text{ else } e | e \ e & \text{expressions}
\end{align*}
\]

Here the predecessor and successor operations do not have their own syntax. Instead, there are two “built-in” constants by the names of \text{pred} and \text{succ}. These constants denote functions that—when applied (using the usual application form \(e_1 e_2\))—produce the predecessor or successor of their arguments.

Questions:

(a) Show the typing rules for this language.

(b) Show the small-step evaluation rules for the language. (You only have to show the rules for function application. Make sure you show all the rules that deal with function application.)

(c) State the canonical forms lemma for this language. (You do not need to prove it.)

(d) Explain the proof of the function application case of the progress lemma for this language. In particular, contrast this with your answer to question 2c.