1. Assuming a type \texttt{nat}, the following ML datatype lets us represent binary
   trees that carry natural numbers at their leaves:
   \begin{verbatim}
datatype tree = Leaf of nat | Branch of tree * tree
   \end{verbatim}
   (a) Devise an encoding of such a type in the polymorphic \(\lambda\)-calculus.
       (Your solution should be inspired by the encoding of \texttt{list} as shown
       in Section 6.4 of the Lecture Notes.)
   (b) Let us call the type that is the answer to the previous question
       \texttt{tree}. Show the corresponding \(\lambda\)-terms for the constructors \texttt{Leaf}
       and \texttt{Branch}. (\texttt{Leaf} should have type \texttt{nat \rightarrow tree}, and \texttt{Branch}
       should have type \texttt{tree \rightarrow tree \rightarrow tree}.)
   (c) Show a \(\lambda\)-term (call it \texttt{sumtree}) of type \texttt{tree \rightarrow nat} that, when
       applied to an instance of a \texttt{tree} computes the sum of its leaves.

2. Using the encoding for \texttt{nat} shown in Section 6.4 of the Lecture Notes,
   define a \(\lambda\)-term (call it \texttt{fac}) of type \texttt{nat \rightarrow nat} that computes the factorial
   of its argument.
   For this, remember that the version of the Polymorphic \(\lambda\)-calculus that
   we are using does not have general recursion: there is no \texttt{letrec} or \texttt{fun},
   and there are no recursive types. However, computing the factorial of \texttt{n}
   does not require general recursion. It is sufficient to \textit{iterate} a certain
   function \texttt{n} times. The Church-encoding of a natural number \texttt{n}
   represents precisely the ability of iterating \textit{any} function whose argument- and result
   types match \texttt{n} times.
   Therefore, the task is to find a suitable function to be iterated, to find
   the appropriate initial argument for it, and to find a way of extracting the
   factorial from its result.
   \textit{Hint:} You probably want to use other encoded types as part of your
   solution. In particular, product types (\textit{a.k.a.} pairs) look promising!

3. (\textit{extra credit}) Explain how function \texttt{pred} as shown in the subsection on
   “Naturals” in Section 6.4 of the Lecture Notes works.
   \textit{Hints:} The basic idea is to have a “nonstandard” representation of the
   integers greater than or equal to \(-1\) \((-1, 0, 1, 2, 3, 4, \ldots\)) together with a
   “nonstandard” version of the successor (called \texttt{psucc}) function that works
   on those numbers. Your solution should involve finding answers to all of
   the following questions:
• What is the representation of $-1$?
• What is the representation of $n$ where $n \geq 0$?
• How does $psucc$ perform its job?
• How is $psucc$ being used within the definition of $pred$?