The purpose of this assignment is to reinforce material from the PL tutorial.

Question 1. Convert the following two *MiniLang* expressions written in the concrete syntax to its corresponding first-order abstract syntax:

```
let days be 7 in 3 + days
'ab' ^ 'cd'
```

Question 2. Write the type derivation trees for the two expressions from question 1 according to the static semantic rules.

Question 3. Evaluate (reduce) the two expressions from question 1 according to the dynamic semantic rules.

Question 4. Provide formal definitions of the functions *free*(e) and *bound(e)* that compute the set of free and bound variables, respectively, in *MiniLang* expression e.

Question 5. Convert two expressions from question 1 to its corresponding *higher-order* abstract syntax.

Question 6. Formally define $[e'/x]\,e$ so that e' is substituted only for *free* (not bound) occurrences of x in *MiniLang* expression e, where the abstract syntax of *MiniLang*'s `let` expression is now in higher-order form.

Question 7. The `cadd` function, defined in the Twelf file, `evaluation.elf`, on line 5. What category of functions (lambda abstraction, pi abstraction, etc) does `cadd` belong to?

Question 8. What does "`cadd/z (s z)`" return?

Question 9. The `cadd` function is used to compute the addition of two non-negative integers. Write the Twelf definition of the function `cmult` to compute the product of two non-negative integers.

Question 10. Consider the following extension of *MiniLang* with *primitive recursion* (based on Gödel's System T) and lambda abstractions. [1]

[1]This was created with the help of Robert Harper's notes.

Category	*Item*		*Abstract*	*Concrete*
Types	τ	$::=$	`num`	`num`
		$\mid$	`str`	`str`
		$\mid$	$\texttt{arr}(\tau_1;\ \tau_2)$	$\tau_1 \to \tau_2$
Expression	e	$::=$	x	x
		$\mid$	$\texttt{num}[n]$	n
		$\mid$	$\texttt{str}[s]$	$\texttt{'}s\texttt{'}$
		$\mid$	$\texttt{+}(e_1;\ e_2)$	$e_1 \texttt{ + } e_2$
		$\mid$	$\texttt{\^{}}(e_1;\ e_2)$	$e_1 \texttt{ \^{} } e_2$
		$\mid$	$\texttt{let}(e_1;\ x.e_2)$	$\texttt{let } x \texttt{ be } e_1 \texttt{ in } e_2$
		$\mid$	$\texttt{rec(e;}\ e_0;\ x.y.e_2)$	$\texttt{rec } e\ \{\ \texttt{z} \Rightarrow e_0 \mid\ \texttt{s}(x) \texttt{ with } y \Rightarrow e_1\ \}$
		$\mid$	$\texttt{lam}(\tau;\ x.e)$	$\lambda(x : \tau.e)$
		$\mid$	$\texttt{ap}(e_1;\ e_2)$	$e_1(e_2)$

Figure 1: Grammar of *MiniLang* with Primitive Recursion and Lambda Abstractions

The expression $\texttt{rec(e;}\ e_0;\ x.y.e_2)$ is called primitive recursion. It represents the e-fold iteration of the transformation $x.y.e_1$ starting from e_0, where x is bound to the predecessor of the iteration and y is bound to the result of the x-fold iteration.
Below are the additional rules for the static semantics.

$$\frac{\Gamma \vdash e : \texttt{num} \quad \Gamma \vdash e_0 : \tau \quad \Gamma, x : \texttt{num},\ y : \tau \vdash e_1 : \tau}{\Gamma \vdash \texttt{rec(e;}\ e_0;\ x.y.e_2) : \tau}$$

$$\frac{\Gamma, x : \tau \vdash e : \tau_2}{\Gamma \vdash \texttt{lam}(\tau;\ x.e) : \texttt{arr}(\tau;\ \tau_2)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{arr}(\tau_2;\ \tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{ap}(e_1;\ e_2) : \tau}$$

A lambda abstraction is now a value.

$$\frac{}{\texttt{lam}(\tau;\ x.e)\texttt{value}}$$

Below are the additional rules for the dynamic semantics.

$$\frac{e_1 \mapsto e_1'}{\texttt{ap}(e_1;\ e_2) \mapsto \texttt{ap}(e_1';\ e_2)}$$

$$\frac{}{\texttt{ap}(\texttt{lam}(\tau;\ x.e);\ e_2) \mapsto [e_2/x]e}$$

$$\frac{e \mapsto e'}{\texttt{rec(}e\texttt{;}\ e_0\texttt{;}\ x.y.e_2\texttt{)} \mapsto \texttt{rec(}e'\texttt{;}\ e_0\texttt{;}\ x.y.e_2\texttt{)}}$$

$$\frac{}{\texttt{rec(num[0];}\ e_0\texttt{;}\ x.y.e_2\texttt{)} \mapsto e_0}$$

$$\frac{n > 0}{\texttt{rec(num[}n\texttt{];}\ e_0\texttt{;}\ x.y.e_2\texttt{)} \mapsto \texttt{[}e, \texttt{rec(num[}n-1\texttt{];}\ e_0\texttt{;}\ x.y.e_2\texttt{)}/x, y\texttt{]}\,e_2}$$

Finally, the question. Explain in plain English what the following function does.

$$\lambda(n : \texttt{num}.\texttt{rec}\ n\ \{\ \texttt{z}\ \Rightarrow\ \texttt{0}\ \mid\ \texttt{s(}w\texttt{)}\ \texttt{with}\ v \Rightarrow \texttt{2 +}\ v\ \})$$