1. (a)
\[
\begin{align*}
\emptyset & \vdash 1 : \text{Int} \quad \emptyset & \vdash 2 : \text{Int} \\
\emptyset & \vdash (+ 1 2) : \text{Int} \quad \emptyset & \vdash 3 : \text{Int} \\
\emptyset & \vdash (+ (+ 1 2) 3) : \text{Int}
\end{align*}
\]

(b) This expression is not typable in the empty environment, because \(x\) is free. More formally, any typing tree for this expression would need to have the shape
\[
\begin{align*}
\emptyset & \vdash 2 : \text{Int} \\
\emptyset & \vdash 1 : \text{Int} \\
\emptyset & \vdash (+ x 1) : \text{Int}
\end{align*}
\]
but the \((\text{Var})\) inference is invalid, since \(\emptyset(x)\) is not defined.

(c) The \((\text{While})\) rule is based on the assumption that the final value of a \(\text{while}\) expression is always the integer 0.

\[
\begin{align*}
\emptyset & \vdash 0 : \text{Int} \\
\emptyset & \vdash 1 : \text{Leq} \\
\emptyset & \vdash (\leq 0 1) : \text{Bool} \\
\emptyset & \vdash (\text{let} x (\leq 0 1) (+ (x (+ x 1))) : \text{Int}
\end{align*}
\]

where \(TE_1 = \{x \mapsto \text{Int}\}\).

\[
\begin{align*}
\emptyset & \vdash 0 : \text{Int} \\
\emptyset & \vdash 1 : \text{Leq} \\
\emptyset & \vdash (\leq 0 1) : \text{Bool} \\
\emptyset & \vdash (\text{let} x (\leq 0 1) (+ (x (+ x 1) 3)) : \text{Int}
\end{align*}
\]

where \(TE_1 = \{x \mapsto \text{Bool}\}\).

(e) There is no valid proof tree for this expression because the two arms of the \(\text{if}\) don’t have the same type: \((\leq 1 2)\) is \(\text{Bool}\) but 3 is \(\text{Int}\). Thus, the \((\text{If})\) rule cannot be applied. Note that the expression is untypable even though it would evaluate without any problem under the usual dynamic semantics, since the second arm of the \(\text{if}\) will always be taken—but the type system doesn’t know about this.

2. Here is a suitable rule:
\[
\begin{align*}
TE & \vdash e_1 : t_1 \\
TE & \vdash e_2 : t_2 \\
TE & \vdash (\text{before} e_1 e_2) : t_1
\end{align*}
\]
Note that is very important to insist (in the second hypothesis) that $e_2$ has some type $t_2$ even though we do not care what that type is. It might be tempting to write the rule like this:

$$
\frac{TE \vdash e_1 : t_1}{TE \vdash \text{before } e_1 e_2 : t_1} \quad \text{(Before')}
$$

but this would be a mistake: using this rule, $e_2$ could be completely ill-typed yet we would still say the overall before expression was well-typed.