Suppose the following functions have the types given:

\[
\begin{align*}
\text{frob} & : \text{Int} \to \text{String} \to \text{Bool} \\
\text{wurble} & : \text{Bool} \to \text{Int} \\
\text{map} & : (a \to b) \to [a] \to [b] \\
\text{filter} & : (a \to \text{Bool}) \to [a] \to [a] \\
\text{zipWith} & : (a \to b \to c) \to [a] \to [b] \to [c] \\
(\$) & : (a \to b) \to a \to b
\end{align*}
\]

Give types to each of the following (all considered separately), or write that the definition is ill-typed:

1. \(f \, x = x\)
   \(f::\)

2. \(f \, x \, y = x \, y\)
   \(f::\)

3. \(f \, x \, y = y \, x\)
   \(f::\)

4. \(f \, (x : xs) = x\)
   \(f::\)

5. \(f \, (x : xs) = xs\)
   \(f::\)

6. \(f \, b = \text{if} \, b \, \text{then} \, b \, \text{else} \, b\)
   \(f::\)

7. \(f \, x \, y \, z = x \, (y \, z)\)
   \(f::\)

8. \(f = \text{map} \, \text{wurble}\)
   \(f::\)

9. \(f = \text{map} \, \text{frob}\)
   \(f::\)

10. \(f = \text{filter} \, \text{wurble}\)
    \(f::\)

11. \(f = \text{filter} \, \text{frob}\)
    \(f::\)

12. \(f = \text{filter} \, (\text{wurble} \, \text{False})\)
    \(f::\)
13. \( f = \text{filter} (\text{frob} 5) \)

    \( f :: \)

14. \( f = \text{zipWith} (\$) \)

    \( f :: \)

15. \( f \ x = \text{filter} ((\$) \ x) \)

    \( f :: \)

16. \( f \ x = x \ x \)

    \( f :: \)

We now assume the following definitions:

\[
\begin{align*}
id :: & a \to a \\
id & x = x \\
\text{const} :: & a \to b \to a \\
\text{const} a & = a
\end{align*}
\]

Figure out what the following reduce to, or say that the expression is ill-typed:

17. \( \text{map} \ id \ [1, 2, 3] \)

18. \( \text{map} (\text{const} \ False) \ ['x', 'y', 'z'] \)

19. \( \text{filter} (\text{const} \ False) \ "abc" \)

20. \( \text{filter} \ id \ [True, False, True] \)

21. \( \text{zipWith} \ id \ [id, \not] \ [False, True] \)

22. \( id \ \not \ True \)

23. \( id \ 'x' \)

Rewrite the following definitions into one-liners using \( \text{map} \) and \( \text{filter} \):

24. \( f \ [\] = [] \)

    \( f \ (x : xs) = x + 1 : f \ xs \)

25. \( f \ [\] = [] \)

    \( f \ (x : xs) = \begin{cases} 
even x & = f \ xs \\
\text{otherwise} & = x : f \ xs \end{cases} \)

26. \( f \ [\] = [] \)

    \( f \ (x : xs) = \begin{cases} 
even x & = x \div 2 : f \ xs \\
\text{otherwise} & = f \ xs \end{cases} \)
27. \( f \) is defined as:
\[
 f \left( x : xs \right) = \begin{cases} 
 \text{even } y & : f \left( xs \right) \\
 \text{otherwise} & : f \left( xs \right)
\end{cases}
\]
where
\[
y = x \div 2
\]
Consider the following definitions:
\[
\begin{align*}
\left[ \right] & \mathbin{+} ys = ys \\
\left( x : xs \right) & \mathbin{+} ys = x : \left( xs \mathbin{+} ys \right) \\
\text{concatMap } \left[ \right] & = \left[ \right] \\
\text{concatMap } f & \left( x : xs \right) = f \times \mathbin{+} \text{concatMap } f xs
\end{align*}
\]

28. \((\mathbin{+})::\):

29. \text{concatMap}:::

30. Use \text{concatMap} to write a function \text{dup} that duplicates every element in a list. That is \text{dup} [ 'x', 'y', 'z' ] evaluates to [ 'x', 'x', 'y', 'y', 'z', 'z' ].