Anonymity, Polymorphism pt.2, and Higher Order

Anonymous Functions/Unnecessary Function Wrapping

Re-write the following functions as val bindings to anonymous functions:

1. fun double x = x * 2;

2. fun identity x = x

3. fun apply_to_five f = f 5;

Re-write the following expressions without unnecessary “wrapping”:

1. if e then true else false →

2. fn x => f x →

Polymorphic Datatypes

Consider the following datatype binding that represents a binary tree:

```plaintext
datatype ('a, 'b) tree = Leaf of 'a | Node of 'b * ('a, 'b) tree * ('a, 'b) tree
```

- What expressions could this datatype support, and what are their types? List at least 3 here:

- What expressions does this datatype not support, and what are their types? List at least 3 here:
Higher Order Functions

Write the function definition for the following functions:
(Hint: which of map, filter, and fold could be useful here? Any previous function can be used?)

1. double_all which has type \(\text{int list} \rightarrow \text{int list}\). This takes an int list and returns an int list whose elements are twice the original.

2. Write a function join with type \(\text{\texttt{a list list}} \rightarrow \text{\texttt{a list}}\) using fold which returns the concatenation of each element in its argument.

3. count_zeros which has type \(\text{int list} \rightarrow \text{int}\). This takes an int list and returns the number of times "0" appears.

4. Consider the following definitions (from HW1):

\[
\text{type date} = \text{int} \times \text{int} \times \text{int} \\
\text{fun day (d : date) = } \#1 \ d \\
\text{fun month (d : date) = } \#2 \ d \\
\text{fun year (d : date) = } \#3 \ d
\]

Write a function number_in_month whose type is \(\text{\texttt{(a * \text{b} * \text{c}) list}} \times \text{\texttt{b}} \rightarrow \text{\texttt{bool}}\). This takes a list of dates and a month and returns the number of dates that are in the given month.

5. Write a function flat_map which has type \(\text{\texttt{(a -> b list)} * \text{\texttt{a list}} \rightarrow \text{\texttt{b list}}}\). This function should take a function as its first argument which maps elements of the second argument to lists, and then flat_map should return the concatenation of those lists.
(hint: does this sound familiar?)