Anonymous Functions/Unnecessary Function Wrapping

Q1: Rewrite the following functions as val bindings to anonymous functions:

1. fun double x = x * 2;

2. fun identity x =

3. fun apply_to_five f = f 5;

Q2: Rewrite the following expressions without unnecessary “wrapping”:

1. if e then true else false →

2. fn x => f x →

Polymorphic Datatypes

Q3: Consider the following datatype binding that represents a binary tree:

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

Q4: Consider the following code:

```ML
fun fold l f a = 
  case l of 
    [] => a 
  | h::t => f (fold t f a, h)
```

a. What is its type?

b. In what order does it process its elements?

Q5: 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 `int list -> 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 `'a list list -> 'a list` using `fold` which returns the concatenation of each element in its argument.

3. `count_zeros` which has type `int list -> int`. This takes an int list and returns the number of times “0” appears.
4. Consider the following definitions (from HW1):

```haskell
  type date = int * int * int
  fun day (d : date) = #1 d
  fun month (d : date) = #2 d
  fun year (d : date) = #3 d
```

Write a function `number_in_month` whose type is `(a * 'b * 'c) list * 'b -> int`. This takes a list of dates and a month and returns the number of dates in that month.

5. Write a function `flat_map` which has type `(a -> 'b list) * 'a list -> '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?)