Haskell problems

You are adviced to solve these problems with Hugs or ghc. Solutions will be provided.

The last part of the Prelude contains all the predefined functions. Consult it for lots of examples. On efd you will find it as `/usr/local/cs/fp/hugs/share/hugs/lib/Prelude.hs`.

1. Define a function that swaps the elements of a pair, `swap(1,True) => (True,1)`. Check the type!
2. Define a function that returns the last element of a list, `last' [1,2,3] => 3`. See the Prelude for a solution.
3. Define a function that "zips" two lists, `zip' [1,2] [3,4] => [(1,3), (2,4)]`. See the Prelude.
4. Define a function `filter' :: (a -> Bool) -> [a] -> [a]` such that `filter' even [1,2,4,5] => [2,4]` with list comprehension. The function `filter` and `even` are defined in the Prelude.
5. Do the same thing without list comprehension.
6. The function `map` is defined in Hudak. Use list comprehension to define. See the Prelude.
7. What is the type of `map map`? Try to infer the type before asking hugs. Apply it to some arguments.
8. Define a data type to represent arithmetic expressions with +, -, * and / and Double constants.
9. Define a function that evaluates such expressions.
10. Define a function `toString::Expr -> String` that returns a string representation of such expressions. Use parentheses around all composite subexpressions.
11. An integer list is either empty or has a head that is an `Integer` and a tail that is an integer list. Define a data type to represent such lists (without using `[ ]` lists).
12. Solve the same problem for lists that may have elements of any type. All elements of a single list are assumed to be of the same type.
13. The following is a data type for representing a binary tree with values at nodes and leaves.

    ```
    data Tree a = Leaf a | Node (Tree a) a (Tree a)
    ```

    Define a function that returns a "mirror image" of such a tree. `mirror (Node (Leaf 1) 2 (Node (Leaf 3) 4 (Leaf 5))) => Node (Node (Leaf 5) 4 (Leaf 3)) 2 (Leaf 1)`.
14. Define a function `map'` that takes a function and such a tree as arguments and returns a tree where the function has been applied to all the values.
    `map' (+1) (Node (Leaf 1) 2 (Node (Leaf 3) 4 (Leaf 5))) =>`
    `(Node (Leaf 2) 3 (Node (Leaf 4) 5 (Leaf 6)))`. `(+1)` is a function that adds 1 to its argument.