

Functional Programming — Mini-Exercises

CSE 505, Autumn 2001

These are a number of very small exercises that we'll do and discuss in class. You don't need to hand in anything.

1. The `map` function can be defined as follows:

```
map :: (a -> b) -> [a] -> [b]
map f [] = []
map f (a:x) = f a : map f x
```

Write a function `map2` that is analogous to `map` but works for functions of two arguments rather than one. What is its type? For example,

```
map2 (+) [1,2,3] [10,11,12]
```

should evaluate to `[11,13,15]`

2. Consider the following Haskell code.

```
do putStr "please enter some text:"
   s <- readLn
   putStr s
```

Rewrite this using only the primitive operators `>>` and `>>=`.

3. The monadic function `putChar`, defined in the Prelude, has the following type:

```
Char -> IO ()
```

Is the following expression type-correct? What does it do?

```
map putChar "hello world"
```

4. Write a Haskell function `repeatLine` that reads in a line of text, and returns that text concatenated with itself. The type of this function should be

```
repeatLine :: IO String
```

5. If I declare a new type as follows

```
data MyList a = Cons a (MyList a) | Nil
```

What is the type of `(Cons 4 (Cons 3 Nil))`?

What gets printed when you evaluate the expression? How can you get it to print correctly?