LUND UNIVERSITY
Department of Computer Science

Functional programming
17 October 2005, 8-13

Exam

Mark each answer with your initials. Write clearly and comment what you do, that might give you points even if the result is wrong.

1. What is the type of the expression:

```
map (const (++))
```

2. Haskell uses so called "lazy evaluation". What is that? Explain as detailed as you can what its benefits are. Also explain why this feature is missing in most other programming languages.

3. In the list library the function `unfoldr` is defined as:

```
unfoldr :: (b -> Maybe (a,b)) -> b -> [a]
unfoldr f b = case f b of
   Nothing -> []
   Just (a,b) -> a : unfoldr f b
```

With a suitable function `g` it is possible to implement the prelude function

```
iterate :: (a->a) -> a -> [a]
```

as:

```
iterate = unfoldr . g
```

Define the function `g`.

4. Assume we are developing a library for image processing. We might then represent an image as a function from the unit square [0,1]x[0,1] to some collor type. Slightly generalized this may be expressed as:

```
type Image a = Position -> a
type Position = (Float, Float)
```

We may now for example define:

```
type Region = Image Boolean
type ColorImage = Image Color
```

a) Write a function

```
paste :: Region -> Image a -> Image a -> Image a
paste reg im1 im2
```

which pastes `im1` into `im2` wherever `reg` is true.

b) Implement the following functions which convert ordinary funktions to functions on images.

```
lift0 :: a -> Image a
lift1 :: (a -> b) -> Image a -> Image b
lift2 :: (a -> b -> c) -> Image a -> Image b -> Image c
```

so that it, for example, is possible to express the difference between to images as:

```
im1 ‘lift2 (-)‘ im2
```

b) Describe what you need to do in order to be able to write the difference as:

```
im1 - im2
```

5. The type class `Functor` defines a generalisation of the list function `map`:

```
fmap :: Functor a => (b->c) -> a b -> a c
```

If m is a monad, show how `fmap f m` can be implemented as a do-expression.

6. Consider the following function:

```
q [] = []
q (x:xs) = x : q (filter (/=x) xs)
```

a) What is the type of `q` and what does it do?

b) Define `f` and `z` so that `q` can be written:

```
q = foldr f z
```