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. a) Explain what the following function does

```
f = filter . flip elem
```

b) Give an alternative definition of `f` which uses a *list comprehension*.

2. Functions in Haskell are defined in so called *curried form*. Explain what that means and give the rationale for this choice.

3. What is the type of `e` defined below? Motivate your answer.

```
e k = do
   x <- k
   Nothing
   return False
```

4. Give the types for the following operator expressions:

a) The Haskell smiley: `(8-)`
b) Haskell goggles: `(+0).(0+)`
c) Haskell wheels: `(.)(.)`
d) The Haskell monster: `(:[])`
e) A Haskell treasure: `(($)$($))`
f) Haskell swearing: `([]>>=)(\_->[(>=)])`

5. In Haskore music is represented by the data type

```
data Music = Note Pitch Dur [NoteAttribute]   -- a note \ atomic
           | Rest Dur                         -- a rest /    objects
           | Music :+: Music                  -- sequential composition
           | Music :=: Music                  -- parallel composition
           | ...
```

There is also a function which combines notes to lines of music:

```
line = foldr (:+:) (Rest 0) :: [Music] -> Music
```

as well as the reverse function

```
lineToList :: Music -> [Music]
lineToList n@(Rest 0) = []
lineToList (n :+: ns) = n : lineToList ns
```

a) Let `m1` och `m2` be defined as

```
m1 = [Note (C,5) dur :+: Note (D,5) dur, Note (E,5) dur]
m2 = [Note (C,5) dur, Note (D,5) dur, Note (E,5) dur]
```

Explain how the values of the expressions `line m1` and `line m2` differ.

b) Define a function `line2` so that `line2 m1` and `line2 m2` is the same as `line m2`.

c) Define a function `lineToList2` so that `lineToList2 (line m1)` and `lineToList2 (line m2)` is the same as `lineToList (line m2)`.

6. Give an alternative but equivalent definition to the prelude function

```
map :: (a -> b) -> [a] -> [b]
```

by using the function `foldr` so that the definition has the form

```
map f = foldr ...
```