Prüfung Funktionale Programmierung

Datum	:	16.07.2014, 12:30 Uhr
Bearbeitungszeit	:	90 Minuten
Prüfer	:	Prof. Dr. Oliver Braun
Hilfmittel	:	Keine
Erreichbare Punkte	:	90

Name: _____

Vorname: _____

Matrikelnummer: _____ Studiengruppe: _____

Hörsaal: _____ Platz Nr.: _____

Unterschrift: _____

Bitte kontrollieren Sie, ob Sie eine vollständige Angabe mit 5 Aufgaben auf 6 Seiten erhalten haben.

Aufgabe	1	2	3	4	5	Summe
max. Punkte	16	18	18	20	18	90

Anmerkungen:

- Sie müssen als Antworten **keine** kompletten Programme schreiben, sondern nur den explizit verlangten Teil eines Programms.
- Schreiben Sie die Lösungen in die dafür vorgesehenen Kästchen. Sollte Ihnen der Platz dabei nicht reichen, benutzen Sie die Rückseite **und vermerken Sie das im dazugehörigen Kästchen!**
- Die Rückseiten können Sie ansonsten für Nebenrechnungen etc. nutzen.

Aufgabe 1 (16 Punkte)

Ergänzen Sie die Typsignaturen für die folgenden Funktionen:

Hinweis: Denken Sie daran, dass in Haskell Literale überladen sein können, z.B.
`1 :: Num a => a`

(a) (4)

```
fun1 ::
fun1 x y z = z ++ [ a + length b | a <- x, b <- y ]
```

(b) (4)

```
fun2 ::
fun2 a b = do
    input <- readLn :: IO Double
    let a = 1.2
    print $ input + a + b
```

(c) (4)

```
import Control.Applicative ((<*>))
```

```
fun3 ::
fun3 a b = case a of
    Nothing -> error "a"
    _       -> a <*> b
```

(d) (4)

```
fun4 ::
fun4 a b = filter a $ map b $ replicate 12 12
```

Aufgabe 2 (18 Punkte)

Definieren Sie die folgenden Funktionen:

(a) Eine Funktion die eine Liste der ersten fünf Zahlen die größer als 20 sind aus einer gegebenen Liste berechnet. (5)

```
fiveOverTwenty :: [Integer] -> [Integer]
```

(b) Eine Funktion die das Maximum zweier Maybe-Werte so berechnet, dass es kein Maximum gibt, wenn einer der beiden `Nothing` ist. (5)

Tipp: Sie können `max` und die Tatsache nutzen, dass `Maybe` eine Instanz von `Applicative` ist. Sie können es aber auch ohne das lösen.

```
maybeMax :: Ord a => Maybe a -> Maybe a -> Maybe a
```

(c) Eine Funktion die die längere von zwei Listen zurück gibt. Die Funktion soll auch dann zum Ergebnis kommen, wenn eine der beiden Listen unendlich ist. D.h. die Funktion `length` kann nicht genutzt werden. Die Funktion muss nicht funktionieren, wenn beide Listen unendlich sind. (8)

```
longerList :: Eq a => [a] -> [a] -> [a]
```

Aufgabe 3 (18 Punkte)

Gegeben sei folgender Datentyp:

```
data Game a = Running a | GameOver
```

für eine Spiel-Monade. Solange das Spiel läuft, sollen `Running`-Werte miteinander verknüpft werden können. Der Wert `GameOver` dient sowohl als Spielende-Zustand als auch als Fehlerzustand. (Tipp: Denken Sie an `Maybe`.)

(a) Geben Sie für den Datentyp `Game` eine Instanz der Klasse `Functor` an. (6)

```
class Functor f where
    fmap :: (a -> b) -> f a -> f b
```

(b) Geben Sie für den Datentyp `Game` eine Instanz der Klasse `Applicative` an. (6)

```
class (Functor f) => Applicative f where
    pure :: a -> f a
    (<*>) :: f (a -> b) -> f a -> f b
```

(c) Geben Sie für den Datentyp `Game` eine Instanz der Klasse `Monad` an. (6)

```
class Monad m where
    return :: a -> m a
    (>>=) :: m a -> (a -> m b) -> m b
    (>>) :: m a -> m b -> m b
    x >> y = x >>= \_ -> y
    fail :: String -> m a
    fail msg = error msg
```

Aufgabe 4 (20 Punkte)

Geben Sie das Ergebnis der folgenden Ausdrücke an. Sie können Ihre Nebenrechnungen im entsprechenden Kästchen machen. Unterstreichen Sie in dem Fall die Lösung.

(a) `foldr (+) 0 [-1,0..4]` (2)

(b) `filter (<20) . map (12-) . filter (>20) . map (+12) $ [1..10]` (6)

(c) `foldr (:) [] [ x+y | x<-[5,7,6], y<-take 1 $ repeat 1]` (6)

(d) `zipWith (<*>) (repeat $ Just (*5)) $ map Just [1..3]` (6)

Aufgabe 5 (18 Punkte)

Gegeben seien folgender Datentyp und folgende Typsynonyme für Filme:

```
type Title = String
data Movie = Movie
  { title :: Title
  , isRent :: Bool
  }
type Serial = Integer
type Movies = [(Serial, Movie)]
```

Gehen Sie im folgenden davon aus, dass die Seriennummer eindeutig ist und jeder Titel nur einmal in der Filmliste vorkommt.

(a) Definieren Sie eine Funktion (8)

`rentable :: Title -> Movies -> Bool`

die `True` genau dann als Ergebnis hat, wenn der übergebene Title in der Filmliste vorhanden und nicht ausgeliehen ist. Sonst ist das Ergebnis `False`.

(b) Definieren Sie eine Funktion (10)

`rent :: Title -> Movies -> (Maybe Serial, Movies)`

zum Ausleihen eines Filmes. Ist der Film nicht vorhanden oder bereits verliehen, ist die erste Komponente des Ergebnisses `Nothing`, sonst die Seriennummer. Zweite Komponente ist die (evtl. veränderte) Filmliste.