

Prüfung Funktionale Programmierung

Datum	:	23.07.2015, 10:30 Uhr
Bearbeitungszeit	:	90 Minuten
Prüfer	:	Prof. Dr. Oliver Braun
Hilfsmittel	:	Keine
Erreichbare Punkte	:	93

Name: _____

Vorname: _____

Matrikelnummer: _____ Studiengruppe: _____

Hörsaal: _____ Platz Nr.: _____

Unterschrift: _____

Bitte kontrollieren Sie, ob Sie eine vollständige Angabe mit 5 Aufgaben auf 7 Seiten erhalten haben.

Aufgabe	1	2	3	4	5	Summe
max. Punkte	16	18	21	20	18	93

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) fun1 :: (4)
fun1 x y z = z : map show [a + b | a <- x, b <- y, a <= b]

(b) fun2 :: (4)
fun2 a b = do
 input <- readLn :: IO Double
 let a = 1.2
 print \$ input + a + b

(c) import Control.Applicative ((<\$>), (<*>)) (4)

```
fun3 ::  
fun3 a b = case a of  
  Nothing -> a  
  _       -> (++) <$> a <*> Just b
```

(d) fun4 :: (4)
fun4 a b = filter b \$ map a \$ replicate 12 \$ show 12

Aufgabe 2 (18 Punkte)

Definieren Sie die folgenden Funktionen:

- (a) Eine Funktion die berechnet ob in einer Liste nur ungerade Zahlen sind. (5)

```
onlyOdds :: [Integer] -> Bool
```

- (b) Eine Funktion die das Maximum zweier Werte so berechnet, dass es kein Maximum gibt, wenn die beiden Werte gleich sind. (5)

```
maybeMax :: Ord a => a -> 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` darf nicht genutzt werden. Die Funktion muss nicht funktionieren, wenn beide Listen unendlich sind. (8)

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

Aufgabe 3 (21 Punkte)

Gegeben sei folgender Datentyp:

```
data Games a = Running [a] | GameOver
```

für eine Spiele-Monade. Solange alle Spiele laufen, sollen **Running**-Werte miteinander verknüpft werden können. Der Wert **GameOver** dient sowohl als Spielende-Zustand als auch als Fehlerzustand. (Tipps: Denken Sie an **Maybe** und nutzen Sie aus, dass Listen auch Funktoren, applikative Funktoren und Monaden sind).

- (a) Geben Sie für den Datentyp **Games** eine Instanz der Klasse **Functor** an.

(7)

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

- (b) Geben Sie für den Datentyp **Games** eine Instanz der Klasse **Applicative** an.

(7)

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

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

(7)

```
class Applicative m => 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
  fail msg = error msg
```

Sie können für die Definition von `bind` die folgende Funktion verwenden:

```
flattenGames :: [Games a] -> Games a
flattenGames (GameOver : _) = GameOver
flattenGames (game : []) = game
flattenGames (Running xs : games) = case flattenGames games of
  GameOver -> GameOver
  Running xss -> Running $ xs ++ xss
```

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 [1,3..10]`

(2)

(b) `map (flip ($) 12) [(+x) | x <- [1..5]]`

(6)

(c) `filter (<1000) $ take 8000 $ map ((*200) . (+2)) [1..]`

(6)

(d) `length $ filter id $ map (True||) $ concat $ replicate 10 [True, False]`

(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.