technische universität
dortmund

Matrikel – Nr.:

Bitte tragen Sie ihre Matrikelnummer und ihren Namen in die dafür vorgesehenen Felder ein. Bitte in deutlicher Handschrift mit einem schwarzen Stift (nicht Bleistift)
Das Feld mit dem **Barcode ist unbedingt frei zu lassen.**
Danke.

Vorname:

Nachname:

Der Bereich unterhalb dieser Linie kann von der Fakultät frei gestaltet werden.

Midterm Test zur Vorlesung

Funktionale Programmierung

13.06.2025

Aufgabe	1	2	3	Σ
Punkte	10	10	10	30
erreicht				

In dem Test sind insgesamt **30 Punkte** erreichbar.
Zum Bestehen sind mindestens **15 Punkte** (50 %) erforderlich.

Wir wünschen viel Erfolg!

Nach der Korrektur wird Ihr Ergebnis unter einem dreistelligen Pseudonym-Code veröffentlicht, den Sie von der Aufsicht erhalten. Tragen Sie diesen Code hier ein:

Name, Vorname, Matrikelnummer | Bitte unbedingt leserlich ausfüllen!

Aufgabe 1 (Listen) $(5 + 5 = 10$ Punkte)

1. Definieren Sie mittels Listenkomprehension eine Haskell-Liste `teiler :: [(Int, Int)]`, die genau diejenigen Paare enthält, deren erste Projektion eine positive ganze Zahl $n > 0$ ist und deren zweite Projektion ein positiver ganzzahliger Teiler t von n ist (`n `mod` t` ist `0`).

2. Definieren Sie eine Haskell-Funktion mit der Signatur

   `split :: [(a,b)] -> ([a], [b])`,

   die eine Liste von Paaren in das Paar der Listen der ersten und zweiten Projektionen abbildet.

   Beispielaufruf:

   `split [(1,'a'), (2, 'b'), (3, 'c')]`

   evaluiert zu

   `([1,2,3],"abc")`.

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Aufgabe 2 (Bäume) $(4 + 3 + 3 = 10$ Punkte)

Gegeben sei der folgende Haskell-Code:

```
1 data BBaum a = Leer | Blatt a | Knoten (BBaum a) (BBaum a) deriving Show
```

1. Definieren Sie eine Haskell-Funktion mit der Signatur

   `foldBBaum :: b -> (a -> b) -> (b -> b -> b) -> BBaum a -> b`,

   die der aus der Vorlesung bekannten Semantik einer Faltung des Datentypen `BBaum` entspricht.

2. Machen Sie `BBaum` auf sinnvolle Weise zu einer Instanz der Typklasse **`Functor`**.

3. Definieren Sie eine Haskell-Funktion mit der Signatur

   `toBBaum :: [a] -> BBaum a`,

   die eine Liste auf einen `BBaum` abbildet.

   Hierbei soll die leere Liste auf den leeren `BBaum` abgebildet werden. Jede nicht leere Liste soll auf einen `BBaum` abgebildet werden, der den **`head`** der Liste im linken Teilbaum enthält und den `BBaum`, der der Abbildung des **`tail`** der Liste entspricht, im rechten Teilbaum.

   Nutzen Sie zur Definition die Faltung **`foldr`** auf sinnvolle Weise.

   Beispielaufruf:

   `toBBaum [1,2,3]`

   evaluiert zu

   `Knoten (Blatt 1) (Knoten (Blatt 2) (Knoten (Blatt 3) Leer))`.

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Aufgabe 3 (Monaden) $(5 + 5 = 10$ Punkte)

Gegeben sei der aus Vorlesung und Übung bekannte Haskell-Datentyp `State`, sowie seine Instanzen der Typklassen **`Functor`**, `Applicative` und **`Monad`**.

```
newtype State s a = State {runS :: s -> (a, s)}

instance Functor (State s) where
    fmap :: (a -> b) -> State s a -> State s b
    fmap f (State g) = State $ \s1 ->
        let (a, s2) = g s1 in (f a, s2)

instance Applicative (State s) where
    pure :: a -> State s a
    pure x = State $ \s -> (x, s)
    (<*>) :: State s (a -> b) -> State s a -> State s b
    (State f) <*> (State h) = State $ \s1 ->
        let (g, s2) = f s1 in let (a, s3) = h s2 in (g a, s3)

instance Monad (State s) where
    (>>=) :: State s a -> (a -> State s b) -> State s b
    (State f) >>= g = State $ \s1 ->
        let (a, s2) = f s1 in let state = g a in runS state s2
```

1. Gegeben sei die folgende Haskell-Funktion:

   ```
   fiMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
   fiMap p f ls = [f x | x <- ls, p x]
   ```

   Übersetzen Sie die Definition von `fiMap` in die `(>>=)`-Notation als Haskell-Funktion mit folgender Signatur:

   `fiMapB :: (a -> Bool) -> (a -> b) -> [a] -> [b]`.

   Sie dürfen die Funktion **`guard`** aus `Control.Monad` für diese Aufgabe nutzen.

2. Gegeben sei der folgende Haskell-Code zur Modellierung eines endlichen Automaten in Haskell.

```haskell
data Zustand = Warte | Bereit deriving Show

data Ausgabe = PinEingeben | Fehler | OK deriving Show

karte :: Zustand -> (Ausgabe, Zustand)
karte _ = (PinEingeben, Bereit)

pin :: Int -> Zustand -> (Ausgabe, Zustand)
pin n Bereit | n == 1234 = (OK, Warte)
             | otherwise = (Fehler, Bereit)
pin _ Warte = (Fehler, Warte)

dau :: Zustand -> ([Ausgabe], Zustand)
dau s0 = let
    (a1, s1) = pin 1111 s0
    (a2, s2) = karte s1
    (a3, s3) = pin 1234 s2
    in ([a1, a2, a3], s3)

karteM :: State Zustand Ausgabe
karteM = State karte

pinM :: Int -> State Zustand Ausgabe
pinM = State . pin
```

Definieren Sie die Funktion `dau :: Zustand -> ([Ausgabe], Zustand)` erneut als Instanz der Zustandsmonade

`dauM :: State Zustand [Ausgabe]`.

Nutzen Sie zur Definition von `dauM` die `do`-Notation auf sinnvolle Weise. Der Konstruktor `State` darf zur Definition von `dauM` nicht verwendet werden.

Beispielaufruf:

```
runS dauM Warte
```

evaluiert zu

```
([Fehler,PinEingeben,OK],Warte)
```

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!

Name, Vorname, Matrikelnummer Bitte unbedingt leserlich ausfüllen!