

Einführung in die Informatik 2

Name	Vorname	Studiengang	Matrikelnummer
.....	<input type="checkbox"/> Bachelor <input type="checkbox"/> Inform. <input type="checkbox"/> Master <input type="checkbox"/> W-Inf. <input type="checkbox"/> <input type="checkbox"/>
Hörsaal	Reihe	Sitzplatz	Unterschrift
.....

Allgemeine Hinweise

- Bitte füllen Sie obige Felder in Druckbuchstaben aus und unterschreiben Sie!
- Bitte schreiben Sie nicht mit Bleistift oder in roter/grüner Farbe!
- Die Arbeitszeit beträgt 120 Minuten.
- Alle Antworten sind in die geheftete Angabe auf den jeweiligen Seiten (bzw. Rückseiten) der betreffenden Aufgaben einzutragen. Auf dem Schmierblattbogen können Sie Nebenrechnungen machen. Der Schmierblattbogen muss ebenfalls abgegeben werden, wird aber in der Regel nicht bewertet.
- Es sind keine Hilfsmittel außer einem DIN-A4-Blatt zugelassen.

Hörsaal verlassen von bis / von bis

Vorzeitig abgegeben um

Besondere Bemerkungen:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	Σ	Korrektor
Erstkorrektur											
Zweitkorrektur											

Aufgabe 1 (5 Punkte)

Geben Sie den allgemeinsten Typ der folgenden Ausdrücke an:

1. `map length`
2. `[] : ([] : [])`
3. `\f g x -> f (g x)`
4. `map (filter fst)`

Begründen Sie kurz, warum der folgende Ausdruck nicht typkorrekt ist:

5. `\b -> if b then b + b else b`
-

Aufgabe 2 (6 Punkte)

1. Implementieren Sie die folgenden Funktionen rekursiv, ohne die Verwendung von Listenkomprehensionen und Funktionen höherer Ordnung wie `any`, `map`, `filter`, `foldl`, `foldr`, `zip`.
 - (a) Die Funktion `exists :: (a -> Bool) -> [a] -> Bool` bekommt ein Prädikat p und eine Liste xs als Argumente und gibt genau dann `True` zurück, wenn ein Element x in xs existiert, für das $p\ x == \text{True}$ ist.
 - (b) Die Funktion `index :: [a] -> [(Int, a)]` bildet eine Liste $[x_0, x_1, \dots, x_n]$ auf $[(0, x_0), (1, x_1), \dots, (n, x_n)]$ ab.
2. Implementieren Sie die Funktion `exists` aus der ersten Teilaufgabe mit Hilfe von `foldr`, ohne die Verwendung von Listenkomprehensionen, Rekursion oder Funktionen höherer Ordnung außer `foldr` (wie `any`, `map`, `filter`, `foldl`, `zip`).

Zur Erinnerung: Die Definition von `foldr` ist:

```
foldr :: (a -> b -> b) -> b -> [a] -> b
foldr f s [] = s
foldr f s (x : xs) = f x (foldr f s xs)
```

Aufgabe 3 (6 Punkte)

Gegeben sei folgender Datentyp zur Darstellung arithmetischer Ausdrücke:

```
data AExp = Cst Int | Var String | Sum AExp AExp | Prod [AExp]
  deriving (Eq, Show)
```

Ein arithmetischer Ausdruck ist entweder eine numerische Konstante, eine Variable, eine binäre Summe oder ein n -stelliges Produkt ($n \geq 0$).

Beispiel: `Prod [Cst 2, Var "y", Sum (Cst 3) (Cst 4)]` stellt $2 \times y \times (3 + 4)$ dar.

Implementieren Sie eine Funktion `simplify :: AExp -> AExp`, die die folgenden Regeln so lange anwendet, bis der Ausdruck (mit diesen Regeln) nicht weiter vereinfacht werden kann:

```
Sum (Cst i) (Cst j) ~> Cst (i + j)
Sum (Cst 0) e ~> e
Sum e (Cst 0) ~> e
```

Beispiel:

```
simplify (Sum (Var "x") (Sum (Cst 1) (Cst (-1)))) == Var "x"
```

Aufgabe 4 (5 Punkte)

Gegeben sei ein Datentyp zur Darstellung binärer Zahlen (Bits):

```
data Bit = Bit0 | Bit1
  deriving (Eq, Show)
```

Die grundlegenden mathematischen Operationen sind in der vereinfachten Typklasse Num definiert:

```
class Num a where
  (+)          :: a -> a -> a
  (*)          :: a -> a -> a
  fromInteger :: Integer -> a
```

Damit sich die Bits nicht benachteiligt fühlen, sollen die üblichen mathematischen Operatoren auch mit Bits funktionieren. Der folgende Code registriert den Datentyp `Bit` als eine Instanz der Typklasse `Num`. Füllen Sie die Lücken so aus, dass `+` und `*` Modulo-2-Arithmetik implementieren. (In Modulo-2-Arithmetik sind alle geraden bzw. ungeraden Zahlen gleich. Zum Beispiel: $1 + 1 = 2 = 0$.)

```
... where
  ...
  fromInteger n = if even n then Bit0 else Bit1
```

Aufgabe 5 (7 Punkte)

Gegeben seien die folgenden Definitionen:

```
data Tree a = Tip | Nd a (Tree a) (Tree a)

mapT :: (a -> b) -> Tree a -> Tree b
mapT f Tip = Tip
mapT f (Nd x t1 t2) = Nd (f x) (mapT f t1) (mapT f t2)
```

Beweisen Sie, dass

$$\text{mapT } g \text{ (mapT } f \text{ t)} = \text{mapT } (g \text{ . } f) \text{ t}$$

Aufgabe 6 (4 Punkte)

Gegeben sei eine Funktion $g :: [\text{Float}] \rightarrow [\text{Float}]$, die eine Liste $[x_1, \dots, x_n]$ von Zahlen auf die Liste $[1/x_1, \dots, 1/x_n]$ abbildet. Das Verhalten der Funktion für Eingabelisten, die die Zahl 0 enthalten, ist nicht spezifiziert.

Beispiel: $g [2.0, 4.0, 0.5] == [0.5, 0.25, 2.0]$.

Erstellen Sie eine Testsuite aus bis zu zwei QuickCheck-Tests, die das Verhalten von g vollständig beschreibt.

Aufgabe 7 (5 Punkte)

Werten Sie die folgenden Ausdrücke Schritt für Schritt mit Haskell's Reduktionsstrategie vollständig aus:

1. `f ([1] ++ [xx]) []`

2. `g 42 43`

3. `h h`

wobei

```
f :: [a] -> [b] -> Bool
f (x:xs) ys = False
f xs      [] = True
```

```
(++) :: [a] -> [a] -> [a]
[]    ++ ys = ys
(x:xs) ++ ys = x : (xs ++ ys)
```

```
g :: Num a => a -> a -> a
g b = \a -> 1*a - b
```

```
h :: a -> Bool
h g = True
```

```
xx :: Num a => a
xx = xx + 0
```

Unendlich lange Reduktionen bitte mit „...“ abbrechen, sobald Nichtterminierung erkennbar ist.

Aufgabe 8 (5 Punkte)

Implementieren Sie die Funktion `remdups :: Eq a => [a] -> [a]`, die alle Duplikate aus einer Liste entfernt. Das erste Vorkommen jedes Elements soll erhalten bleiben. Beispielsweise soll gelten `remdups [1,5,3,1,0,3] == [1,5,3,0]`.

Die Funktionen `nub` und `nubBy` aus der Haskell-Standardbibliothek dürfen nicht verwendet werden.

Aufgabe 9 (7 Punkte)

Definieren Sie eine IO-Aktion `ssp :: IO ()`, die das Spiel Stein-Schere-Papier realisiert. In jeder Runde wählen Benutzer und KI (Computer) jeweils einen Wert vom Typ `Waffe`.

```
data Waffe = Stein | Schere | Papier
```

Die Benutzereingabe wird mit `readLn :: Read a => IO a` eingelesen, den Zug der KI würfelt `randomIO :: Random a => IO a` aus. Verwenden Sie die vordefinierte Funktion `schlaegt :: Waffe -> Waffe -> Bool` um den Sieger einer Runde zu bestimmen: Ein Spieler gewinnt die Runde und bekommt einen Punkt, wenn seine Waffe v die Waffe w seines Gegners schlägt (`schlaegt v w`). Für Unentschieden und Niederlage gibt es keinen Punkt. Sie dürfen annehmen, dass `Waffe` eine Instanz von `Eq`, `Show`, `Read` und `Random` ist.

Ihr Programm soll bei korrekten Eingaben nicht terminieren und nach jeder Eingabe die zufällig generierte Waffe der KI, das Ergebnis der aktuellen Runde, sowie den Spielstand ausgeben (mit `putStrLn :: String -> IO ()` und `show :: Show a => a -> String`). Beispiel (Benutzereingaben sind *kursiv* dargestellt):

```
Stein
```

```
vs. Papier. KI gewinnt! Stand: KI 1:0 Sie
```

```
Schere
```

```
vs. Schere. Unentschieden! Stand: KI 1:0 Sie
```

```
Schere
```

```
vs. Papier. Sie gewinnen! Stand: KI 1:1 Sie
```
