Prüfung vom 4. November 2019

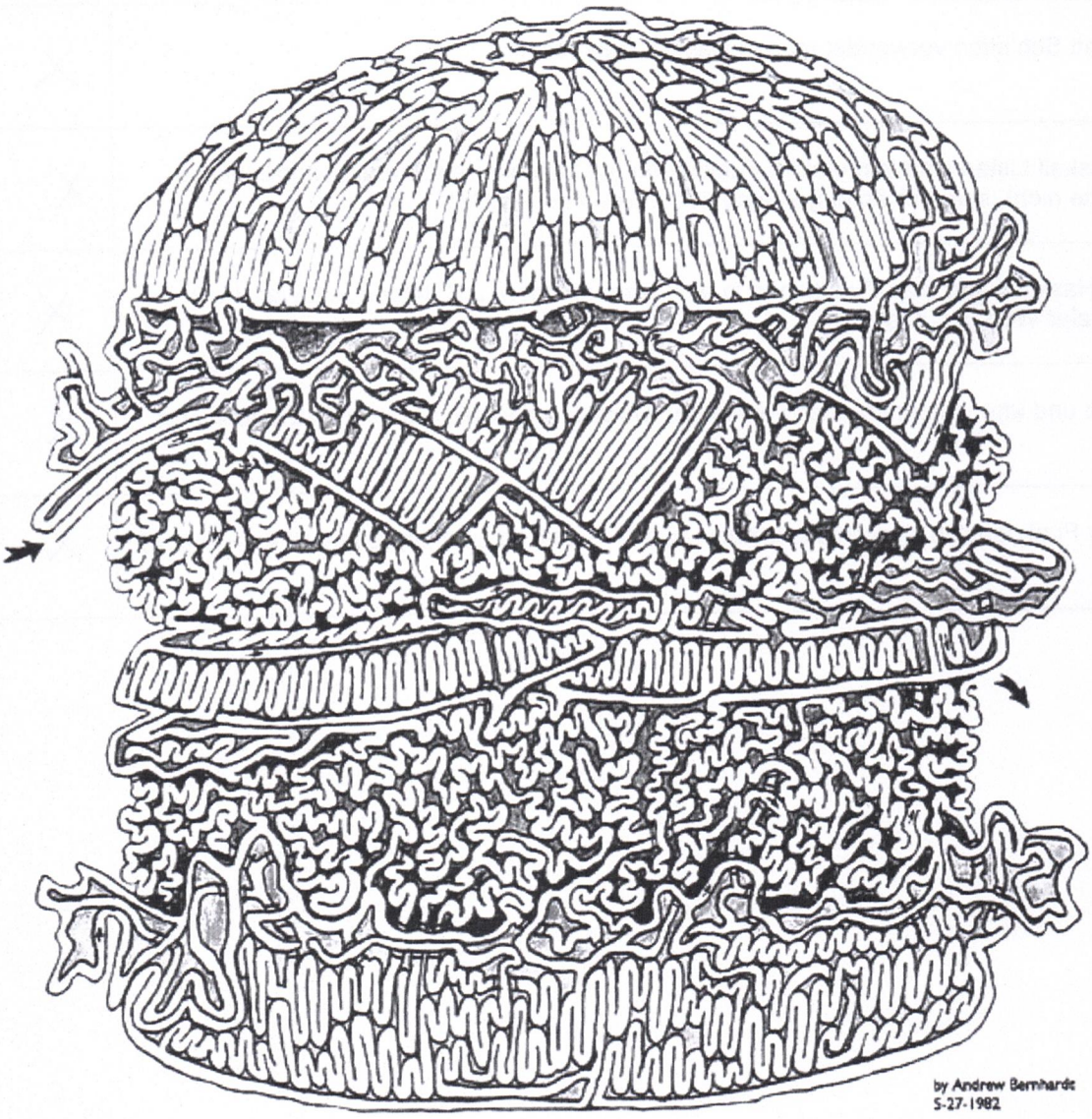
Name, Vorname: Merkli Nico

Allgemeine Hinweise:

1) Schreiben Sie Ihren Namen auf diese Blatt.
2) Sie dürfen ein beidseitig beschriebenes/bedrucktes DIN-A4 Blatt mit Notizen verwenden. Sonst ist ausser einem Stift nichts erlaubt.
3) Bitten beantworten Sie die Fragen direkt auf den Aufgabenblättern. Verwenden Sie allenfalls die leere Rückseite.
4) Lesen Sie die Prüfung zuerst durch und beginnen Sie mit den Aufgaben, die Ihnen am meisten liegen.
5) Sie haben 45 Minuten Zeit.
6) Bleiben Sie bitte Ruhig an Ihrem Platz bis die Zeit um ist.

Viel Erfolg!

*Falls Sie frühzeitig fertig sind, können Sie sich gerne mit folgendem Labyrinth beschäftigen. Es ist **nicht** Notenrelevant.*



4

Aufgabe 1: Functional Programming Basics (4 Punkte)

Kreuzen Sie für alle folgenden Aussagen entweder *Richtig* oder *Falsch* an. Aussagen ohne Kreuz, mit zwei Kreuzen oder unklar angekreuzte Felder werden neutral mit 0 Punkten bewertet. Lesen Sie die Aussagen genau durch!

(0.5 Punkte pro richtige Antwort, *0.5 Punkte Abzug pro falsche Antwort*, min. 0 Punkte)

Aussage	Richtig	Falsch
Haskell Funktionen geben für gleiche Argumente immer dasselbe Resultat zurück.	X	
Tuples in Haskell sind homogen: Jede Komponente muss denselben Typ haben.		X
In Haskell können nur Werte mit dem gleichen Typ mittels == verglichen werden.	X	
Statt Schleifen verwendet man in Haskell Rekursion.	X	
Haskell Listen sind immutable. Funktionen auf Listen verändern die ursprüngliche Liste nicht, sondern erzeugen eine neue Liste als Resultat.	X	
In Haskell kann man Funktionen in eine Liste ablegen. Z.B. folgende Liste ist ein legaler Wert [fst, snd].	X	
`let` und `where` können auch verwendet werden, um lokale Funktionen zu definieren.	X	
Der Funktionstyp ist rechtsassoziativ: a -> b -> c ist so zu interpretieren: a -> (b -> c)	X	

12

Aufgabe 2: Typen **(6 * 2 = 12 Punkte)**

Gesucht ist der Typ des jeweiligen Ausdrucks. Verwenden Sie `Int` für allfällige numerische Typen.
Wenn Sie keine oder mehrere Felder ankreuzen, gilt die Aufgabe als falsch, genauso unklar angekreuzte Felder. (2 Punkte pro korrekte Antwort)

Gegeben sind folgende Definitionen:

```
a :: (a, Int) -> [a]
b :: (Int -> Char) -> Bool
c :: [String] -> Bool
fst :: (a,b) -> a
```

```
head :: [a] -> a
map :: (a -> b) -> [a] -> [b]
filter :: (a -> Bool) -> [a] -> [a]
(.) :: (b -> c) -> (a -> b) -> a -> c
```

`head [(1," "),(2,"B")]`
☐ `[(Int,String)]`
☒ `(Int,String)`
☐ `(Int,Char)`
☐ Ungültiger Ausdruck

`c.a`
☐ `(a,Int) -> Bool`
☐ `[String] -> [a]`
☒ `(String,Int) -> Bool`
☐ Ungültiger Ausdruck

`map (\(x:_) -> x)`
☐ `[a] -> [a]`
☐ `[[a]]`
☒ `[[a]] -> [a]`
☐ Ungültiger Ausdruck

`fst.(\(e,f,g) -> g)`
☒ `(w,x,(y,z)) -> y`
☐ `((x,y,z),w) -> x`
☐ `(a,b,c) -> c -> (a,b) -> a`
☐ Ungültiger Ausdruck

`filter snd`
☐ `[(a,b)] -> [(a,b)]`
☒ `[(a,Bool)] -> [(a,Bool)]`
☐ `((a,b) -> b) -> [b]`
☐ Ungültiger Ausdruck

`map b []`
☐ `[(Int -> Char)] -> [Bool]`
☐ `[(Int -> Char)] -> Bool`
☒ `[Bool]`
☐ Ungültiger Ausdruck

6

Aufgabe 3: Werte **(6 * 1 = 6 Punkte)**

Gesucht ist jeweils der Wert von a. Schreiben Sie den Wert auf die dafür vorgesehene Linie.
(1 Punkt pro Aufgabe)

a)

```
f x | x > 1 = [x]
    | otherwise = []

a = map f [2,1]
```

Wert von a : [[2], []] ✓

b)

```
f = filter (\(_:as) -> length as >= 3)

a = f ["H", "all", "o"]
```

Wert von a : [] ✓

c)

```
f = head.tail.tail.fst

a = f ("abcd","efgh")
```

Wert von a : 'c' ✓

d)

```
f 0 = 1
f n = f (n-1)

a = (f 42) + 1
```

Wert von a : 2 ✓

e)

```
f i = g (i-1)
  where g = (\j -> j < i)

a = f 3
```

2 → innen True

Wert von a : True ✓

f)

```
f [[z1,z2]] = []
f [x:xs]    = xs

a = f [[1,2,3]]
```

✓

Wert von a : [2,3]

3

Aufgabe 4: Generische Funktion **(3 Punkte)**

Implementieren Sie eine Funktion mit folgender Typsignatur:

```
f :: a -> ((a -> b) -> c) -> (a -> a -> b) -> c
```

Die zu definierende Funktion muss legal terminieren. D.h. die Verwendung der Funktionen `error` und `undefined` sowie die Verwendung von Rekursion die nicht abbricht, ist nicht erlaubt. Ebenso nicht erlaubt sind vordefinierte Funktionen. Verwenden Sie allenfalls Patternmatching.

Hinweise:
- Es gibt nur eine einzige Lösung. Schauen Sie genau auf die Typen und überlegen Sie sich, was als Parameter reinkommt und was Sie damit tun können.
- Der Funktionstyp -> assoziiert nach rechts.

```
f a fc fb = fc (fb a)
```

Aufgabe 5: Bibliothek 9 **(1 + 4 + 4 = 9 Punkte)**

In dieser Aufgabe implementieren Sie Funktionen für Bibliothek-Daten. Die Bibliothek führt Bücher und Filme:

```
data Type = Book | Movie deriving Eq
```

Ein ausleihbares Medium besteht aus einem Titel (String), einem Type und einem Mindestalter (Int).

```
type Medium = (String, Type, Int)
```

Die Bibliothek hält alle verwalteten Medien in einer Liste. Hier ein Beispiel:

```
media :: [Medium]
media = [ ("Haskell Programming", Book, 12),
          ("The return of Haskell", Movie, 18),
          ("Bambi", Movie, 4)]
```

Aufgaben:

Eigene rekursive Implementierungen sind in dieser Aufgabe nicht erlaubt!

1 a) Implementieren Sie die Funktion ageLimit, die das Mindestalter aus einem Medium extrahiert:
ageLimit :: Medium -> Int
Beispiel:
ageLimit ("The return of Haskell", Movie, 18) == 18

```
ageLimit (_,_,a) = a
```

4 b) Gesucht ist die Funktion mediaByAge die alle Medien, die für ein gegebenes Alter zugelassen sind, zurückgibt:
[Medium] → [Medium]
mediaByAge:: Int -> ~~[Media] -> [Media]~~

Beispiel:
mediaByAge 13 media == [("Haskell Programming", Book, 12), ("Bambi", Movie, 4)]
Verwenden Sie die in Aufgabe a) definierte Funktion ageLimit um das Mindestalter zu extrahieren.

```
~~mediaByAge a m = filter (\x -> x<=~~
~~mediaByAge a m = filter (\x -> x<=a)~~
mediaByAge a m = filter (\x -> (ageLimit x)<=a) m
```

4 c) Gesucht ist die Funktion titles, die bei gegebenem Mindestalter und einer Liste von Medien eine Liste der Titel zurückgibt:

```
titles :: Int -> [Media] -> [String]
```

Beispiel:

```
titles 13 media == ["Haskell Programming", "Bambi"]
```

Verwenden Sie die in Aufgabe b) definierte Funktion um alle zugelassenen Medien für das gegebene Alter zu finden. Verwenden Sie ein Pattern im Lambda-Parameter um den Titel zu extrahieren.

```
~~titles a m = map (\(_,s,_) -> s)~~
titles a m = map (\(s,_,_) -> s) (mediaByAge a m)
```

11

Aufgabe 6: Rekursion **(6 + 6 = 12 Punkte)**

In dieser Aufgabe programmieren Sie selbst rekursive Funktionen.

Die Verwendung von vordefinierten Listenfunktionen ist in dieser Aufgabe nicht erlaubt. Verwenden Sie Patternmatching um die Listen zu zerlegen. head, tail und length sind ebenfalls nicht erlaubt.

5 a) Gesucht ist die rekursive Implementierung der Funktion `duplicateAt`. Sie nimmt ein Index vom Typ `Int` und eine Liste mit Elementtyp a und verdoppelt das Element an der Indexposition. Der Index ist 0-basiert. Die Funktion hat folgende Signatur:

```
duplicateAt :: Int -> [a] -> [a]
```

Beispiele:

```
duplicateAt  1 [2,1,4] == [2,1,1,4]
duplicateAt  2 ['a','b','c','d'] == ['a','b','c','c','d']
```

Sollte der Index ausserhalb der Liste liegen, wird die Liste unverändert zurückgegeben. Negative Indizes können Sie in dieser Aufgabe ignorieren.

```
duplicateAt _ [] = []
```

```
duplicateAt 0 (x:xs) = x:x:xs
duplicateAt i (x:xs)
  | i>(length xs) = (x:xs)
  | otherwise     = x:(duplicateAt (i-1) xs)
```

Alternative:

```
duplicateAt 0 (x:xs) = x:x:xs
duplicateAt i (x:xs) = if (i > (length xs)) then (x:xs) else x:(duplicateAt (i-1) xs)
```

6 b) Gesucht ist die rekursive Implementierung der Funktion `findGt`. Die Funktion hat folgende Typsignatur:

```
findGt :: Int -> [Int] -> Int
```

Mit dieser Funktion findet man den ersten Wert in einer Liste, der grösser ("Greater than") als der erste Parameter ist. Falls es keinen Wert in der Liste hat, der grösser als der erste Parameter ist, soll 0 zurückgegeben werden.

Beispiele:

```
findGt 3 [2,4,3] == 4 -- 4 ist der erste Wert in der Liste der grösser als 3 ist
findGt 8 [2,3,4] == 0 -- kein Wert ist grösser als 8
```

Verwenden Sie Patternmatching und Guards für die Fallunterscheidung!

```
findGt _ [] = 0
```

```
findGt i (x:xs)
  | i<x = x
  | otherwise = findGt i xs
```